
*Analisi dell'impatto di errori in un dataset su
tre modelli di Machine learning*

By: Matteo Varè 856479

Sommario

Obiettivo elaborato.....	3
I modelli d'addestramento utilizzati	3
Gaussian Naive-Bayes.....	3
Decision Tree.....	4
Neural network	4
Il Dataset	5
Analisi e manipolazioni effettuate sul dataset originale per migliorare l'addestramento	5
Il processo di sporcatura.....	9
Def main(...)	9
Def runTestset(...)	10
Def downsampling(...)	14
Def prepareData(...)	15
Def sporcaX(...)	15
Inconsistenza nei dati	16
Valori Mancanti nel Dataset.....	18
Ripetizioni di voci nel dataset	20
Errori di Selezione in un menu a tendina (o simili).....	21
Errori di input manuali.....	23
Def fitEncoding(...)	26
Def evaluate_model(...)	27
Esecuzione test ed Analisi risultati.....	29
Caso pulito.....	29
Inconsistenza dei dati 10%,90%.....	36
Valori Mancanti nel Dataset 10%-50%,70%.....	49
Inserimento valori NaN 25%,45%	83
Ripetizioni di voci nel dataset 30%-80%-999%	87
Errori di Selezione 20%-40%	105
Errori di input manuali 20%,80%	123
Conclusioni	136

Obiettivo elaborato

L'obiettivo dell'elaborato è quello di analizzare come alcune tipologie di sporcature in un dataset possano influenzare il training di modelli di machine-learning.

Per fare ciò sono stati effettuati i seguenti macro-passaggi:

- È stato preso un "dataset" assunto pulito da kaggle.com
- Il dataset è stato analizzato e adattato alle necessità di input degli algoritmi di Machine-Learning utilizzati
- È stato sporcato il train-set in maniera controllata ed incrementale
- Ad ogni stadio di sporcatura sono stati utilizzati tre algoritmi di Machine-Learning diversi
- Sono stati analizzati i risultati facendo il confronto con i valori forniti in output tra i vari modelli di Machine-Learning

I modelli d'addestramento utilizzati

I modelli utilizzati per l'analisi sono il Naive-Bayes di tipo Gaussiano, l'Albero Decisionale ed il Neural Network.

Di seguito analizzeremo questi tre modelli.

Gaussian Naive-Bayes

La classificazione Bayesiana è un algoritmo di apprendimento supervisionato con il quale si determina la probabilità di un elemento di appartenere a una certa classe.

È stata scelta la variante Gaussiana in quanto le altre non erano coerenti col contenuto del dataset.

Il dataset presenta caratteristiche compatibili con i requisiti più importanti per un apprendimento via Naive-Bayes:

1. Presenza di un dataset di grandi dimensioni (45000 righe per 14 colonne)
2. Il problema affrontato è di classificazione binaria (approvato / non approvato)
3. Esiste una variabile target che facilita l'apprendimento supervisionato

Inoltre, bisogna considerare che il Naive-Bayes è un modello relativamente veloce da addestrare (soprattutto se confrontato con le Support Vector Machines).

Il Naive-Bayes richiede che le features siano indipendenti e che abbiano l'assunzione di normalità . Per ovviare a ciò sono state applicate delle modifiche al dataset (di cui parleremo in seguito).

Decision Tree

L'albero decisionale è un algoritmo di apprendimento supervisionato e parametrico.

Presenta una struttura gerarchica ad albero che consiste in un singolo nodo radice, molteplici rami, nodi interni e foglie.

È un algoritmo di apprendimento supervisionato molto flessibile (non richiede l'indipendenza tra le Features) e facile da interpretare. In teoria riesce anche a gestire le variabili categoriche.

Tuttavia, nel nostro lavoro tali variabili sono state convertite in numeriche (con metodi che vedremo più avanti) perché come specificato nella documentazione di scikit-learn, l'algoritmo che useremo non riesce per ora a gestire le suddette variabili categoriche direttamente da valori string.

Neural network

Il modello utilizzato è il Multi-layer Perceptron classifier (MLPClassifier) di scikit-learn.

È un algoritmo di apprendimento supervisionato che apprende una funzione non lineare per la classificazione e utilizza la backpropagation per l'addestramento.

Il modello ottimizza la funzione log-loss utilizzando LBFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) (algoritmo di ottimizzazione) o la discesa del gradiente stocastico.

Il MLPClassifier è capace di modellare relazioni complesse e non lineari e può lavorare bene con feature ridotte ma complesse (utile per l'analisi di alcune features “sporcate”).

Il MLPClassifier, però, richiede scaling dei dati ed è lento da addestrare.

Inoltre, è bene prendere in considerazione che presenta molti iperparametri da regolare e, come molti classificatori neurali, non è interpretabile a causa della natura “a scatola chiusa” dell’addestramento.

Il Dataset

Il dataset è stato ottenuto da kaggle.com, in particolare è <https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data>.

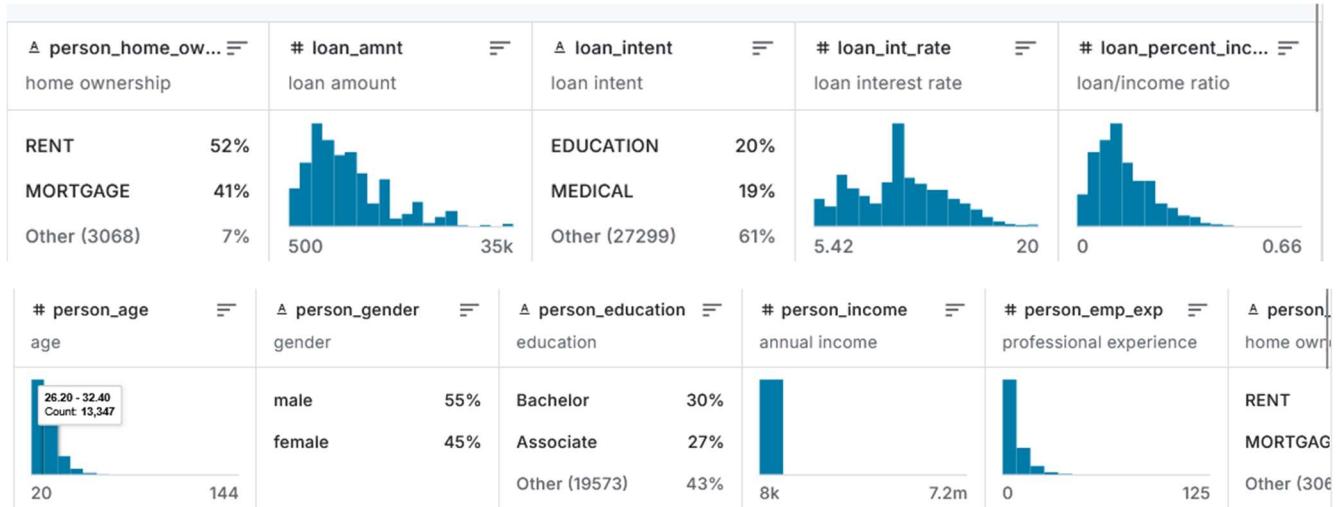
Analisi e manipolazioni effettuate sul dataset originale per migliorare l'addestramento

Prima di tutto è stato necessario analizzare i valori del dataset, per far ciò sono stati utilizzati i grafici già forniti su kaggle e la stampa dei valori contenuti nelle categoriche.

Per la stampa dei valori contenuti nelle categoriche è stato utilizzato il seguente codice:

```
[ ] 1 data.head()
[ ] 2 person_age person_gender person_education person_income person_emp_exp person_home_ownership loan_amnt loan_intent loan_int_rate loan_percent_income cb_person_cred_hist_length credit_score previous_loan_defaults_on_file loan_status
[ ] 3 0 22.0 female Master 71948.0 0 RENT 35000.0 PERSONAL 16.02 0.49 3.0 561 No 1
[ ] 4 1 21.0 female High School 12282.0 0 OWN 1000.0 EDUCATION 11.14 0.08 2.0 504 Yes 0
[ ] 5 2 25.0 female High School 12438.0 3 MORTGAGE 5500.0 MEDICAL 12.87 0.44 3.0 635 No 1
[ ] 6 3 23.0 female Bachelor 79753.0 0 RENT 35000.0 MEDICAL 15.23 0.44 2.0 675 No 1
[ ] 7 4 24.0 male Master 66135.0 1 RENT 35000.0 MEDICAL 14.27 0.53 4.0 586 No 1
[ ] 8
[ ] 9 1 categorical_cols = ['person_gender', 'person_education', 'person_home_ownership', 'loan_intent', 'previous_loan_defaults_on_file'] #categoriche
[ ] 10 2 for col in categorical_cols:
[ ] 11 3 if col in data.columns:
[ ] 12 4     if len(data[col].unique()) == 1:
[ ] 13 5         print(f"(col): {data[col].unique()}")
[ ] 14 6     else:
[ ] 15 7         print(f"Column '{col}' not found in the DataFrame.")
[ ] 16
[ ] 17 person_gender: ['female', 'male']
[ ] 18 person_education: ['Master', 'High School', 'Bachelor', 'Associate', 'Doctorate']
[ ] 19 person_home_ownership: ['RENT', 'OWN', 'MORTGAGE', 'OTHER']
[ ] 20 loan_intent: ['PERSONAL', 'EDUCATION', 'MEDICAL', 'VENTURE', 'HOMEIMPROVEMENT', 'DEBTCONSOLIDATION']
[ ] 21 previous_loan_defaults_on_file: ['No', 'Yes']
```

Di seguito i grafici forniti su kaggle:

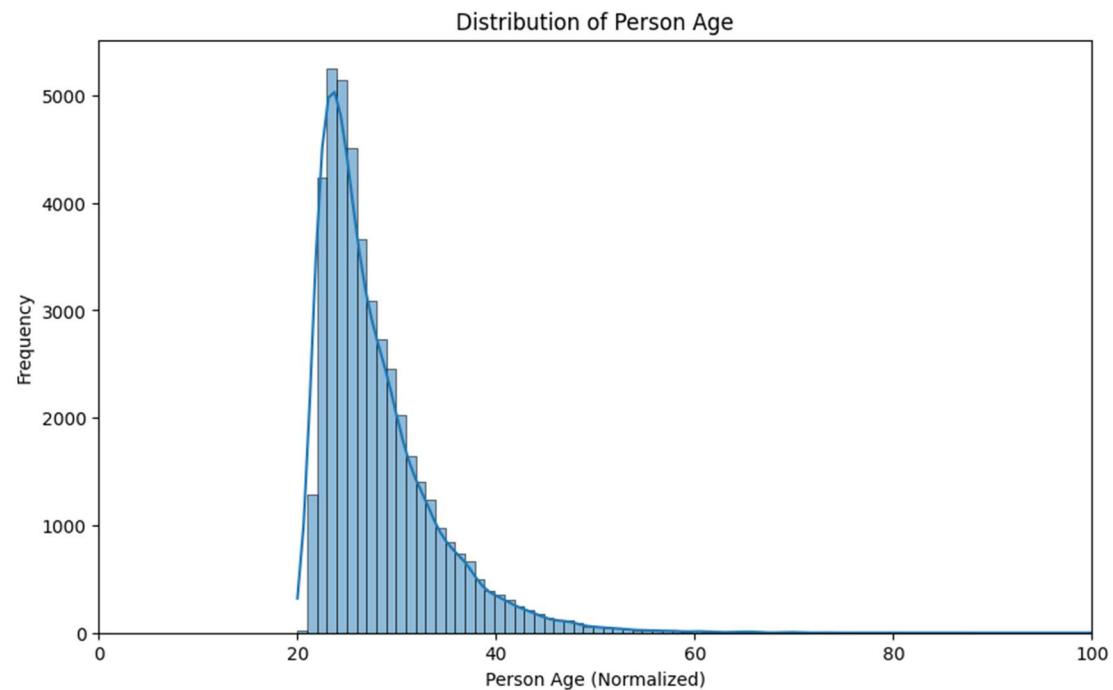
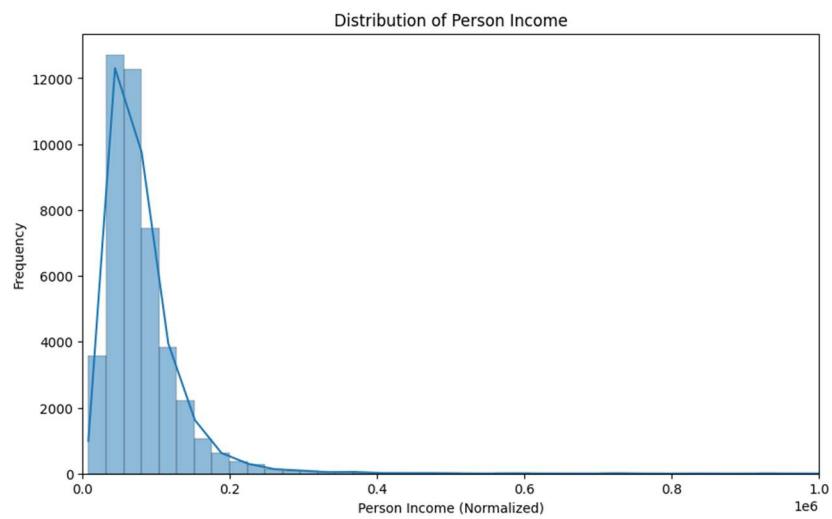


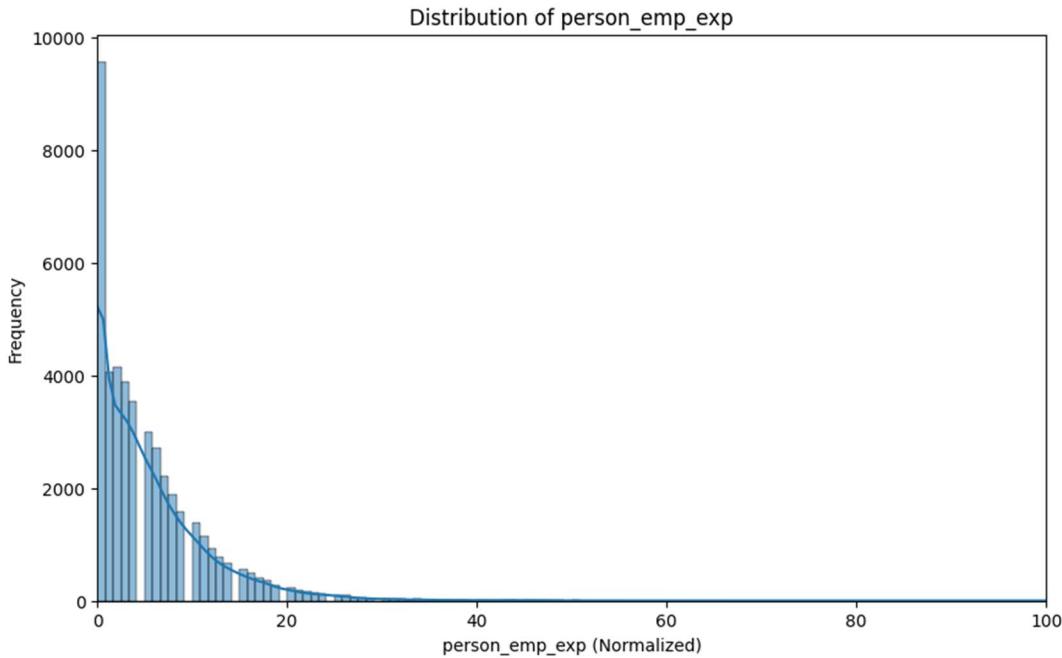
È bene notare che “person_income”, “person_age” e “person_emp_exp” hanno una distribuzione fortemente non normale in questa visualizzazione dei grafici, quindi per sicurezza è stata fatta una seconda visualizzazione a granulatura più piccola per vedere come era effettivamente la distribuzione.

Di seguito un esempio di codice utilizzato

```
1 # Visualize the distribution of 'person_income' using a histogram or bar plot
2 plt.figure(figsize=(10, 6))
3 sns.histplot(data['person_age'], bins=125, kde=True)
4 plt.xlabel('Person Age (Normalized)')
5 plt.ylabel('Frequency')
6 plt.title('Distribution of Person Age')
7 plt.xlim(0, 100)
8 plt.show()
```

Di seguito i risultati ottenuti:





È bene notare che con questa visualizzazione sia “person_income” che ”person_age” risultano avere una distribuzione abbastanza normale mentre “person_emp_exp” presenta un esagerato quantitativo di valori nel primo bin dell’istogramma che possono intaccare la normalità della distribuzione.

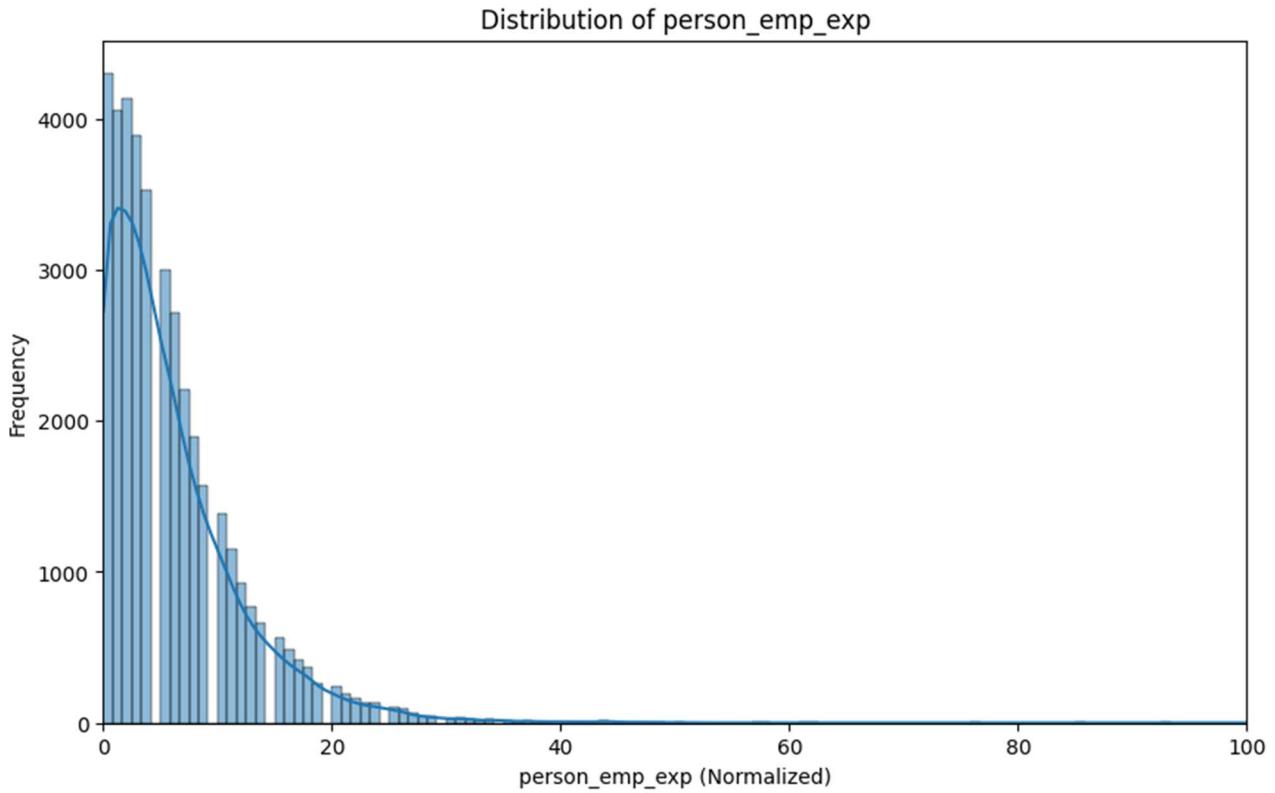
Per ovviare a ciò sono stati eliminati il 55% dei valori (randomicamente) del primo bin con questa funzione:

```

1 def remove_first_bin_values(data, feature_to_remove_from, percentage_to_remove):
2
3     newData = data.copy()
4
5     if feature_to_remove_from in newData.columns:
6         feature_data = newData[feature_to_remove_from]
7
8         # Calculate histogram bins and counts for the specified feature
9         # Use a large number of bins to approximate the first bar as values >= min and < first_bin_end
10        counts, bin_edges = np.histogram(feature_data.dropna(), bins=150) # Calculate bins on non-NaN data
11
12        if len(bin_edges) > 1:
13            first_bin_start = bin_edges[0]
14            first_bin_end = bin_edges[1]
15
16            # Identify rows in the first bin
17            first_bin_indices = newData[(feature_data >= first_bin_start) & (feature_data < first_bin_end)].index.tolist()
18
19            num_to_remove = int(len(first_bin_indices) * percentage_to_remove)
20            num_to_remove = min(num_to_remove, len(first_bin_indices)) # Ensure not removing more than available
21
22            if num_to_remove > 0:
23                indices_to_remove = random.sample(first_bin_indices, num_to_remove)
24                # Remove the entire rows based on the selected indices
25                newData = newData.drop(indices_to_remove)
26                print(f"Rimosso {num_to_remove} ({percentage_to_remove:.2%}) righe dalla prima barra di '{feature_to_remove_from}' .")
27            else:
28                print(f"Nessuna riga rimossa dalla prima barra di '{feature_to_remove_from}' .")
29        else:
30            print(f"Non è stato possibile determinare i bin per la feature '{feature_to_remove_from}'. Nessuna riga rimossa.")
31    else:
32        print(f"La feature '{feature_to_remove_from}' non è presente nel DataFrame.")
33
34    return newData

```

Con la seguente distribuzione finale:



È bene notare che la condizione di normalità della classe considerata è assunta tale nel Naive Bayes Gaussiano solo in linea teorica. Infatti, il GaussianNB può essere applicato e performare bene anche con dati che non sono perfettamente normali.

Altre potenziali non-normalità non così facilmente correggibili sono state lasciate perché non sembravano influire in maniera particolarmente negativa nell'addestramento senza sporcature ed era potenzialmente interessante notare come le sporcature possano influenzare questo aspetto del dataset.

È stata eliminata la feature 'loan_percent_income' perché la sua presenza causava problemi di collinearità.

Inoltre, è stato notato che quando previous_loan_defaults_on_file = "Yes" la target è sempre pari a 0, quindi per avere un miglior addestramento sono state eliminate le entrate con tale condizione e poi l'intera feature è stata droppata.

Di seguito i codici a riguardo:

```
1 if 'previous_loan_defaults_on_file' in data.columns:  
2     initial_rows = len(data)  
3     data = data[data['previous_loan_defaults_on_file'] != 'Yes'].copy() # Keep rows where the value is not 'Yes'  
4     rows_removed = initial_rows - len(data)  
5     print(f"Rimosse {rows_removed} righe dove 'previous_loan_defaults_on_file' era 'Yes'.")  
6     display(data.head())  
7 else:  
8     print("La colonna 'previous_loan_defaults_on_file' non è presente nel DataFrame.")
```

[]	1 data = data.drop('loan_percent_income', axis = 1)
[]	1 data = data.drop('previous_loan_defaults_on_file', axis = 1) 2 display(data.head())
⤵	person_age person_gender person_education person_income person_emp_exp person_home_ownership loan_amnt loan_intent loan_int_rate cb_person_cred_hist_length credit_score loan_status
0	22.0 female Master 71948.0 0 RENT 35000.0 PERSONAL 16.02 3.0 561 1
1	21.0 female High School 12282.0 0 OWN 1000.0 EDUCATION 11.14 2.0 504 0
2	25.0 female High School 12438.0 3 MORTGAGE 5500.0 MEDICAL 12.87 3.0 635 1
3	23.0 female Bachelor 79753.0 0 RENT 35000.0 MEDICAL 15.23 2.0 675 1
4	24.0 male Master 66135.0 1 RENT 35000.0 MEDICAL 14.27 4.0 586 1

Il processo di sporcatura

Il testo seguirà l'ordine di esecuzione delle funzioni del programma una volta eseguita la chiamata; quindi, partendo da “def main” si concluderà con “def evaluate_model”.

Def main(...)

Di seguito riporto la funzione “main” del programma.

```
1 def main(olddata,inconsistenza=0,nan=0,ripetizioni=0,selezioni=0,MIA=0,input_err=0):
2
3     X_test, y_test ,y_train ,modelis,x_train_type = runTestset(olddata.copy(),inconsistenza=inconsistenza,nan=nan,ripetizioni=ripetizioni,selezioni=selezioni,MIA=MIA,input_err=input_err)
4
5     for model in models:
6         current_x_train_info = next(item for item in x_train_type if item["name"] == model["name"])
7         current_x_train = current_x_train_info["x_train"]
8         # Pass X_test and y_test ecc.. from the unpacked values to evaluate_model
9         evaluate_model(model = model['model'], x_train_type = current_x_train,y_train = y_train ,X_test = X_test,y_test= y_test , y_pred=model.get('y_pred'), y_pred_proba=model.get('y_pred_proba'),y_train_pred=model.get('y_train_pred'),y_train_pred_proba=model.get('y_train_pred_proba'))
```

Questa è la funzione principale che viene chiamata ogni volta dal programma quando si vuole inserire un particolare tipo di sporcatura ed analizzarne l'impatto di essa.

Essa è definita con:

```
def main(olddata,inconsistenza=0,nan=0,ripetizioni=0,selezioni=0,MIA=0,input_err=0):
```

“olddata” è il dataset che andremo a sporcare.

inconsistenza, nan, ripetizioni, selezioni, MIA ed input_err sono le tipologie di sporcature da inserire nel dataset. In particolare il valore inserito rappresenterà la percentuale del dataset che verrà sporcato con quel determinato errore.

Le tipologie di errori sono impostati ad un valore di default pari a 0 così da poter eseguire l'analisi del dataset pulito con il comando “main(dataset_pulito)” e poter verificare come ogni singola tipologia di sporcatura possa incidere sul dataset in maniera mirata.

I valori di inconsistenza, nan, ripetizioni, selezioni, MIA ed input_err possono variare da $0 \leq n < 1$.

Il main è composto da due blocchi principali:

1. La chiamata a runTestset(...) che restituisce i valori per: X_test, y_test, y_train, models (array di dizionari), x_train_type(array di dizionari)
2. Un ciclo for che itera sull' array di dizionari di model. All' interno di ogni iterazione viene identificato il valore corretto di x_train nell' array di dizionari di x_train_type relativo al modello analizzato e viene composta una chiamata alla funzione evaluate_model(...) con tutte le informazioni estratte dai due dizionari identificati.

Def runTestset(...)

Questa funzione viene chiamata solo dal def main(...). Essa si occupa sia di alcune operazioni preliminari sul dataset che dell'addestramento degli algoritmi di classificazione.

“runTestset” è definita con:

```
def runTestset(odata,inconsistenza=0,nan=0,ripetizioni=0,selezioni=0,MIA=0,input_err=0):
```

i valori di questi input sono gli stessi degli omonimi specificati nel capitolo precedente “def main(...)”.

Inoltre, questa funzione presenta un output definito con:

```
return X_test, y_test, y_train, models, x_train_type
```

X_test, y_test ed y_train sono alcuni dei valori ottenuti in seguito a prepareData(...).

“Models” e “x_train_type” sono degli array di dizionari.

“Models” contiene dizionari con informazioni riguardo il modello utilizzato e le predizioni degli algoritmi per ogni algoritmo di addestramento.

X_train_type contiene dizionari con informazioni riguardo alla tipologia di x_train utilizzato per un particolare algoritmo di addestramento.

La funzione può essere divisa in tre sottosezioni, in ordine dall'alto:

1. Preparazione del dataset ed inizializzazione delle liste di modelli ed x_train_type

```
4| data = odata.copy()
5| data = downsampling(data)
6|
7| X_train, X_test, y_train, y_test = prepareData(data,inconsistenza=inconsistenza,nan=nan,ripetizioni=ripetizioni,selezioni=selezioni,MIA=MIA,input_err=input_err)
8|
9| # initiate list to store models and predictions
10| models = []
11|
12| # initiate x_train_type to store X_train used
13| x_train_type = []
```

2. Imputazione dei valori Nan (necessario, altrimenti causa errori critici nella fase di addestramento)

```
17 ##### Imputation for NaN values
18
19 # imputer for NaN values
20 imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
21
22 # Fit on training data and transform both training and test data
23 X_train_imputed = imputer.fit_transform(X_train)
24 X_test_imputed = imputer.transform(X_test)
25
26 # Convert back to DataFrame to retain column names
27 # Use the index from the original X_train/X_test to maintain alignment
28 X_train_imputed = pd.DataFrame(X_train_imputed, columns=X_train.columns, index=X_train.index)
29 X_test_imputed = pd.DataFrame(X_test_imputed, columns=X_test.columns, index=X_test.index)
30
```

3. Addestramento dei vari modelli di classificazione

```
33 ##### naive-bayes
34 gnb = GaussianNB()
35 # Use the imputed data for training and prediction
36 gnb.fit(X_train_imputed, y_train)
37
38 y_pred_gnb = gnb.predict(X_test_imputed)
39 y_train_pred_gnb = gnb.predict(X_train_imputed)
40
41
42 # GaussianNB might raise errors or behave unexpectedly with NaNs if not handled -> imputed NaNs
43 models.append({
44     'name': 'Naive Bayes',
45     'model': gnb,
46     'y_pred': y_pred_gnb,
47     'y_pred_proba': gnb.predict_proba(X_test_imputed)[:, 1], # Get probability for the positive class
48     'y_train_pred': y_train_pred_gnb,
49     'y_train_pred_proba': gnb.predict_proba(X_train_imputed)[:, 1] # Get probability for the positive class
50 })
51
52 x_train_type.append({
53     'name': 'Naive Bayes',
54     'x_train': X_train_imputed
55 })
```

```

58 ##### decision-tree
59
60 dtc = DecisionTreeClassifier(random_state=42, max_depth=6, min_samples_split=3)
61
62 # Use the imputed data for training and prediction
63
64 dtc.fit(X_train_imputed, y_train)
65
66 y_pred_dtc = dtc.predict(X_test_imputed)
67 y_train_pred_dtc = dtc.predict(X_train_imputed)
68
69 # DTC can sometimes handle NaNs, but behavior varies. Used imputed data for consistency.
70 models.append({
71     'name': 'Decision Tree',
72     'model': dtc,
73     'y_pred': y_pred_dtc,
74     'y_pred_proba': dtc.predict_proba(X_test_imputed)[:, 1], # Get probability for the positive class
75     'y_train_pred': y_train_pred_dtc,
76     'y_train_pred_proba': dtc.predict_proba(X_train_imputed)[:, 1] # Get probability for the positive class
77 })
78
79 x_train_type.append({
80     'name': 'Decision Tree',
81     'x_train': X_train_imputed
82 })
83
84 ##### neural network
85
86 # Inizializziamo lo StandardScaler (necessario per il nn)
87 scaler = StandardScaler()
88
89 # Applichiamo lo scaling solo alle feature numeriche
90 # Scaliamo il training set e poi trasformiamo il test set con lo stesso scaler
91 X_train_scaled = scaler.fit_transform(X_train_imputed)
92 X_test_scaled = scaler.transform(X_test_imputed)
93
94 # Creiamo un'istanza della Rete Neurale
95 # hidden_layer_sizes: Una tupla che specifica il numero di neuroni in ogni strato nascosto.
96 #                     in questo caso: un solo strato nascosto con tot neuroni.
97 # activation: Funzione di attivazione per gli strati nascosti ('relu' è il piu' utilizzato).
98 # solver: Algoritmo per l'ottimizzazione dei pesi ('adam' è un default).
99 # max_iter: Numero massimo di epoch.
100 # random_state: Per riproducibilità.
101 # alpha: Termine di regolarizzazione L2 (per prevenire overfitting).
102 # learning_rate_init: Tasso di apprendimento iniziale.
103 nn_model = MLPClassifier(
104     hidden_layer_sizes=(30,), # Un singolo strato nascosto con tot neuroni
105     activation='relu',       # Funzione di attivazione ReLU
106     solver='adam',          # Ottimizzatore Adam
107     max_iter=5000,          # Numero massimo di epoch
108     random_state=42,         # Random state per riproducibilità
109     alpha=0.0001,            # Regolarizzazione L2 (piccola)
110     learning_rate_init=0.01 # Tasso di apprendimento
111 )

```

```

113 # Addestramento della Rete Neurale sul training set scalato
114
115 nn_model.fit(X_train_scaled, y_train)
116
117 y_train_pred_nn = nn_model.predict(X_train_scaled)
118 y_test_pred_nn = nn_model.predict(X_test_scaled)
119
120 # Select only the probability for the positive class (used later in models.append ecc)
121 y_train_pred_proba_nn = nn_model.predict_proba(X_train_scaled)[:, 1]
122 y_test_pred_proba_nn = nn_model.predict_proba(X_test_scaled)[:, 1]
123
124
125 models.append({
126     'name': 'Neural Network',
127     'model': nn_model,
128     'y_pred': y_test_pred_nn,
129     'y_pred_proba': y_test_pred_proba_nn, # Get probability for the positive class
130     'y_train_pred': y_train_pred_nn,
131     'y_train_pred_proba': y_train_pred_proba_nn # Get probability for the positive class
132 })
133
134
135 x_train_type.append({
136     'name': 'Neural Network',
137     'x_train': X_train_imputed
138 })
139

```

Nelle fasi di addestramento è bene evidenziare similitudini e differenze nell’ implementazione dei vari modelli:

in ogni implementazione è stata utilizzata la stessa logica per l’addestramento sul training set, con l’unica differenza che nel neural network non è stato utilizzato il training set imputato ma anche scalato (il perché verrà spiegato successivamente).

Alla fine di ogni addestramento sono state aggiornate le liste di dizionari “models” ed “x_train_type” aggiungendo dei nuovi dizionari contenente le informazioni relative ai modelli ed all’ “x_train” utilizzato.

Nel decision tree è stato scelto di applicare “max_depth=6” e “min_samples_split=3” in quanto valori che riducevano notevolmente gli indicatori di overfitting ed underfitting.

Nel neural network è risultata necessaria un’applicazione sul training set inputato di uno standard scaler.

Ciò è stato effettuato perché le reti neurali sono sensibili alla scala delle feature di input, in particolare, applicare lo StandardScaler sul training set prima dell’addestramento favorisce una convergenza più veloce e più stabile perché le feature ottengono scale simili , inoltre dato che molte funzioni di attivazione sono sensibili all’intervallo dei valori di input, scalando i dati si tende a mantenere i valori in intervalli dove queste funzioni hanno gradienti più significativi, favorendo un apprendimento efficace.

Inoltre, è bene notare che applicare lo StandardScaler su un dataset contenente sia features numeriche che categoriche gestite con l'one-hot encoder non è un problema; infatti, le features “fantoccio” generate con l'one-hot encoder semplicemente finiranno con l'avere due valori distinti: uno negativo ed uno positivo, centrati sullo zero e con una deviazione standard di 1 invece che 0 ed 1.

Def downsampling(...)

Questa funzione viene chiamata solo dal def runTestset (...).

Dato che è stato notata una grande disparità tra i loan_status = 0 ed i loan_status = 1 è stato necessario effettuare il downsampling del dataset.

“downsampling” è definita con:

```
def downsampling(OldData):
```

E ritorna un output definito con

```
return data
```

“OldData” è il dataset che deve subire il downsampling mentre “data” è il nuovo dataset dopo il downsampling

Di seguito il codice della funzione:

```
1 def downsampling(OldData):
2
3     data = OldData.copy()
4
5     # Conta le occorrenze delle classi di loan_status
6     class_counts = data['loan_status'].value_counts()
7
8     # Identifica la classe minoritaria e maggioritaria
9     minority_class = class_counts.idxmin()
10    majority_class = class_counts.idxmax()
11    minority_count = class_counts[minority_class]
12    majority_count = class_counts[majority_class]
13
14    # DataFrame per la classe minoritaria
15    minority_df = data[data['loan_status'] == minority_class]
16
17    # DataFrame per la classe maggioritaria
18    majority_df = data[data['loan_status'] == majority_class]
19
20    # Downsampling della classe maggioritaria per avere lo stesso numero di campioni della classe minoritaria
21    # Manteniamo random_state qui per la riproducibilità della selezione del sottoinsieme
22    majority_downsampled = majority_df.sample(n=minority_count, random_state=420)
23
24    # Concatena i DataFrame bilanciati
25    balanced_df = pd.concat([minority_df, majority_downsampled])
26
27    # Resetta l'indice del DataFrame bilanciato e mescola le righe per la riproducibilità
28    data = balanced_df.sample(frac=1, random_state=420).reset_index(drop=True).copy()
29
30    return data
```

Def prepareData(...)

Questa funzione viene chiamata solo dal def runTestset (...). Essa si occupa di creare lo split del dataset originario in valori di test e valori di train, e sporcare il test-set attraverso “sporcaX”. Infine, applica un “fitEncoding” sui nuovi valori di train e poi lo utilizza sia sui valori sporchi che puliti per creare coerenza nel dataset totale in caso di creazione di nuove colonne categoriche in seguito alla sporcatura.

“prepareData” è definita con:

```
def prepareData(data,inconsistenza=0,nan=0,ripetizioni=0,selezioni=0,MIA=0,input_err=0):
```

I valori di questi input sono gli stessi degli omonimi specificati nel capitolo “def main(...)”.

Inoltre, questa funzione presenta un output definito con:

```
20 |     return X_train, X_test, y_train, y_test
```

Il contenuto di questo output sono i valori che in seguito il programma utilizzerà per addestrare e testare gli algoritmi di machine learning. In particolare X_train e y_train sono valori che variano a seconda del tipo e del livello di sporcatura applicata al dataset.

Di seguito il codice della funzione:

```
1 def prepareData(data,inconsistenza=0,nan=0,ripetizioni=0,selezioni=0,MIA=0,input_err=0):
2
3     newData = data.iloc[:,0:-1].copy() # seleziona tutto tranne l'ultima colonna # usa .copy() per evitare SettingWithCopyWarning
4     target = data.iloc[:, -1].copy() # seleziona solo l'ultima colonna ( target )
5
6     # Assicura che newData e target abbiano lo stesso indice prima dello split
7     newData.reset_index(drop=True, inplace=True)
8     target.reset_index(drop=True, inplace=True)
9
10
11    X_train, X_test, y_train, y_test = train_test_split(newData, target, test_size = 0.30,
12                                                       stratify= target, #mantiene le proporzioni delle classi differenti dato che il dataset potrebbe essere sbilanciato
13                                                       random_state= 420) #fissato random seed per replicabilità
14
15
16    X_train, y_train = sporcaX(X_train,y_train,inconsistenza=inconsistenza,nan=nan,ripetizioni=ripetizioni,selezioni=selezioni,MIA=MIA,input_err=input_err)
17
18    X_train,y_train,X_test,y_test = fitEncoding(X_train,y_train,X_test,y_test) # fitta l'encoder sul train sporco e poi lo utilizza sia per lo sporco che per il pulito per coerenza
19
20    return X_train, X_test, y_train, y_test
```

Def sporcaX(...)

Questa funzione viene chiamata solo dal def prepareData (...). Essa si occupa di sporcare i valori della porzione di dataset utilizzata per l’allenamento degli algoritmi di machine learning.

“sporcaX” è definita con:

```
def sporcaX(X_train, y_train, inconsistenza=0,nan=0,ripetizioni=0,selezioni=0,MIA=0,input_err=0):
```

I valori di X_train ed y_train sono la porzione di dataset finalizzata all’addestramento dei modelli di machine learning. In questo caso i valori sono del dataset pulito.

Gli altri valori di input sono gli stessi degli omonimi specificati nel capitolo “def main(...)”.

Inoltre, questa funzione presenta un output definito con:

```
356 |     return xv_train, yv_train
```

Ove sono contenuti i valori sporcati di X_train ed y_train secondo le specifiche di input.

In seguito ad una preliminare copia dei valori di input X_train ed y_train e la creazione di un dataset unificato, vengono applicate le sporcature secondo le richieste dell' input.

Di seguito il codice per le operazioni preliminari:

```
3 |     xv_train = X_train.copy()
4 |     yv_train = y_train.copy()
5 |
6 |     # unione target al dataset per le sporcature per quando necessario avere entrambe
7 |
8 |     df_unified = pd.concat([xv_train, yv_train], axis=1)
9 |
```

Inconsistenza nei dati

Questo tipo di operazione comporta l'inserimento di inconsistenze logiche tra i valori delle features del dataset.

Per fare ciò sono state necessarie individuare delle features che hanno delle relazioni logiche implicite tra loro, le features così individuate sono:

- Età (person_age) – anni di lavoro (person_emp_exp) in quanto non ha senso avere più anni di lavoro che età
- Età (person_age) – anni di credito (cb_person_cred_hist_length) in quanto non ha senso avere più anni di credito che età

Quindi, per applicare questo tipo di inconsistenze è bastato modificare i valori in person_emp_exp ed in cb_person_cred_hist_length in modo tale che siano maggiori del loro relativo person_age.

Di seguito l'implementazione con eventuali commenti dove necessario:

```
110 |     # Inconsistenze nei dati
111 |     # piu' anni di lavoro (person_emp_exp) che età
112 |     # piu' anni di credito (cb_person_cred_hist_length ) che età
113 |     if inconsistenza > 0:
114 |         num_inconsistent = int(len(xv_train) * 2 * inconsistenza) # max value = numrows*2 ; una riga può contenere tutte e due le incostenze
115 |
116 |         # Colonne tra cui scegliere per le inconsistenze
117 |         inconsistent_cols = ['person_emp_exp', 'cb_person_cred_hist_length']
118 |         # Ottieni gli indici numerici delle colonne da corrompere
119 |         inconsistent_col_indices = [xv_train.columns.get_loc(col) for col in inconsistent_cols]
120 |
121 |         person_emp_exp = 0
122 |         cb_person_cred_hist_length = 0
123 |
```

Qui vengono fatte le operazioni preliminari, notare come è stato deciso che una riga può contenere entrambe le tipologie di inconsistenze e quindi la percentuale degli errori è stata calcolata sul doppio del numero di righe massimo.

```

124 |     # Seleziona casualmente gli indici delle righe e delle colonne
125 |     if num_inconsistent > 0:
126 |         # Creare una lista di tutte le possibili combinazioni (riga, colonna) per le colonne selezionate
127 |         all_cell_indices = [(r, c_idx) for r in range(xv_train.shape[0]) for c_idx in inconsistent_col_indices]
128 |
129 |         # Assicurati di non selezionare più celle di quelle disponibili
130 |         num_inconsistent = min(num_inconsistent, len(all_cell_indices))
131 |         # Seleziona casualmente gli indici delle celle
132 |         inconsistent_cell_indices = random.sample(all_cell_indices, num_inconsistent)
133 |
134 |         for row_idx, col_idx in inconsistent_cell_indices:
135 |             col_name = xv_train.columns[col_idx] # Get column name from index
136 |
137 |             if col_name == 'person_emp_exp':
138 |                 person_emp_exp += 1
139 |                 current_value= xv_train.iloc[row_idx, col_idx] # anni di lavoro
140 |                 new_value = int (current_value + xv_train.iloc[row_idx, col_idx]) # anni di lavoro pari o maggiori dell' età
141 |                 xv_train.iloc[row_idx, col_idx] = new_value
142 |
143 |             elif col_name == 'cb_person_cred_hist_length':
144 |                 cb_person_cred_hist_length += 1
145 |                 current_value= xv_train.iloc[row_idx, col_idx] # anni di credito
146 |                 new_value = float (current_value + xv_train.iloc[row_idx, col_idx]) # anni di credito pari o maggiori dell' età
147 |                 xv_train.iloc[row_idx, col_idx] = new_value
148 |
149 |             else:
150 |                 print(f"valore non gestito:{xv_train.iloc[row_idx, col_idx]} , colonna: {col_name}, tipo: {type(xv_train.iloc[row_idx, col_idx])} ")
151 |
152 |             print('')
153 |             print('inconsistenze in:')
154 |             print(f'person_emp_exp: {person_emp_exp}')
155 |             print(f'cb_person_cred_hist_length: {cb_person_cred_hist_length}')
156 |             print('')

```

Da riga 126 a riga 132 vengono individuati casualmente gli indici delle celle nelle features da modificare.

Da riga 134 a riga 148 vengono applicate le modifiche dei valori di person_emp_exp e di cb_person_cred_hist_length ed aggiornati i contatori di modifiche effettuate.

A riga 150 è presente un comando di “debug”, se il programma funziona come inteso non dovrebbe mai essere eseguito tale comando.

Da riga 152 a riga 156 sono presenti i comandi di “print” dei valori contatori. Questi valori servono solo per avere una rappresentazione della effettiva distribuzione degli errori introdotti sulle features indicate.

Valori Mancanti nel Dataset

Questa operazione di sporcatura consiste nell' eliminazione di valori nel dataset.

L'eliminazione è possibile sostituendo i valori categorici con “None” ed i numerici con “NaN”.

Di seguito l'implementazione con eventuali commenti dove necessario:

```
263 if MIA > 0:
264     #verisone che inserisce MIAs anche nel target, assume che se il target è mancante è posto = 0 (nessun prestito effettuato)
265     # considerare di eliminare la riga dal dataset se viene eliminata una variabile target
266     num_missing_values = int(xv_train.size * MIA)
267     if num_missing_values > 0:
268         # Ensure we don't try to add more missing values than there are cells
269         num_missing_values = min(num_missing_values, xv_train.size)
270         # Use random.sample for efficiency if adding many missing values
271         all_cell_indices = [(r, c) for r in range(xv_train.shape[0]) for c in range(xv_train.shape[1])]
272         missing_cell_indices = random.sample(all_cell_indices, num_missing_values)
273         numerical=0
274         strings=0
275
276         for row_idx, col_idx in missing_cell_indices:
277             col_name = xv_train.columns[col_idx] # Get column name from index
278             if isinstance(xv_train.iloc[row_idx, col_idx], (float, np.floating, int, np.integer)):
279                 numerical +=1
280                 xv_train.iloc[row_idx, col_idx] = np.nan # Valore sentinella per numerici
281             elif isinstance(xv_train.iloc[row_idx, col_idx],(str)):
282                 strings +=1
283                 xv_train.iloc[row_idx, col_idx] = None # Stringa vuota per categorici
284             else:
285                 print(f"valore non gestito:{xv_train.iloc[row_idx, col_idx]} , colonna: {col_name}, tipo: {type(xv_train.iloc[row_idx, col_idx])} ")
286             print('')
287             print('MIA added:')
288             print(f'numerical: {numerical}')
289             print(f'strings: {strings}')
290             print('')
```

Inserimento valori Nan

Questa operazione di sporcatura consiste in inserire valori NaN (Not a Number) nei valori delle features numeriche.

Questa tipologia di sporcatura è una versione più specifica dell'operazione di inserimento “valori mancanti nel dataset”.

Questa decisione di esplorare come solo i valori NaN influenzano l'addestramento degli algoritmi è stata influenzata da due fattori principali:

1. Alcuni algoritmi di addestramento restituivano errori critici durante l'addestramento se il dataset conteneva valori “NaN”, ciò ha richiesto l'imputazione dei valori “NaN” e quindi una potenziale deviazione del comportamento tipico ed aspettato da esso.
2. la maggior parte delle features sono di tipo numerico e quindi è interessante vedere cosa succede se la maggior parte delle features di un dataset ha degli elementi compromessi (e vedere se le features categoriche possono in qualche modo sostituire quelle numeriche in questo addestramento)

Di seguito l'implementazione con eventuali commenti dove necessario:

```
251 | # Inserimento di valori NaN
252 | if nan > 0:
253 |     # Select only numerical columns for NaN injection
254 |     numerical_cols = xv_train.select_dtypes(include=np.number).columns.tolist()
255 |     if numerical_cols: # Check if there are numerical columns
256 |         num_nan_cells = int(len(xv_train) * len(numerical_cols) * nan)
257 |         if num_nan_cells > 0:
258 |             # Ensure indices are within bounds
259 |             num_nan_cells = min(num_nan_cells, len(xv_train) * len(numerical_cols))
260 |             # Select random row indices
261 |             row_indices = np.random.randint(0, len(xv_train), num_nan_cells)
262 |             # Select random column indices from numerical columns
263 |             col_indices_in_numerical = np.random.randint(0, len(numerical_cols), num_nan_cells)
264 |             # Map back to original DataFrame column names
265 |             col_names_to_corrupt = [numerical_cols[i] for i in col_indices_in_numerical]
266 |
267 |             # Apply NaN based on selected row and numerical column names
268 |             for r_idx, col_name in zip(row_indices, col_names_to_corrupt):
269 |                 xv_train.loc[xv_train.index[r_idx], col_name] = np.nan # Use .loc with actual index
270 |
```

Da riga 252 a riga 265 sono presenti istruzioni per identificare le features numeriche, verificare che non ci siano errori di indicizzazione ed estrapolazione delle celle randomicamente selezionate da esse.

Il ciclo for tra riga 268 e riga 269 si occupa di estrarre e “modificare” le celle del dataset precedentemente identificate.

Ripetizioni di voci nel dataset

Questa operazione di sporcatura consiste nel inserire entrate ripetute nel dataset senza aumentare il numero di entrate in esso.

Questa tipologia di errore può per esempio verificarsi (sebbene in volumi più moderati) quando un dataset viene convertito da un formato di rappresentazione più complesso con impaginazione (per esempio un .pdf) ad uno più semplice e sistematico (tipo un .csv). In particolare, i valori in prossimità delle prime/ultime entrate della pagina nel formato complesso hanno la tendenza a moltiplicarsi quando convertiti in formato più semplice.

Per mantenere il numero totale di entrate (e quindi avere una sufficientemente corretta percentuale di duplicati nel dataset) è stato deciso di scegliere randomicamente le righe del dataset da mantenere e poi scegliere randomicamente quali tra quelle righe duplicare.

Di seguito l'implementazione con eventuali commenti dove necessario:

```
314 # Ripetizione di voci del dataset
315 #calcola la percentuale "ripetizioni" di elementi da rimuovere dal dataset
316 # sceglie randomicamente gli elementi da rimuovere e quelli da duplicare
317
318 if ripetizioni > 0:
319     num_duplicate_rows = int(len(df_unified) * ripetizioni) #utilizzo df_unified per gestire anche il target
320
321     if num_duplicate_rows > 0:
322         num_rows_to_keep = len(df_unified) - num_duplicate_rows
323         # Seleziona casualmente gli indici delle righe originali da mantenere
324         original_indices = random.sample(df_unified.index.tolist(), num_rows_to_keep)
325         df_kept = df_unified.loc[original_indices].copy()
326         # Seleziona casualmente gli indici delle righe da duplicare
327         indices_for_duplicates = random.choices(df_kept.index.tolist(), k=num_duplicate_rows)
328         df_duplicates = df_unified.loc[indices_for_duplicates]
329         # Concatena le righe mantenute con i duplicati
330         df_unified = pd.concat([df_kept, df_duplicates], ignore_index=True)
331         # Reset index after concatenation to ensure a clean integer index
332         df_unified.reset_index(drop=True, inplace=True)
333         print(f"Aggiungi {num_duplicate_rows} duplicati. Dimensione dataset finale: {len(df_unified)}")
334         # divide x e y
335         xv_train = df_unified.drop('loan_status', axis=1) # loan status è il nome della target nel nostro dataset
336         yv_train = df_unified['loan_status']
```

Da riga 318 a riga 322 vengono calcolati il numero di righe da mantenere ed il numero di righe da duplicare.

Da riga 323 a riga 328 vengono selezionati casualmente gli indici delle righe originali da mantenere e tra di essi gli indici delle righe da duplicare.

Riga 329-332 vengono concatenati i due sotto-dataset precedentemente creati e vengono reimpostati gli indici.

A riga 333 viene stampato un messaggio di debug.

Riga 335-336 viene separata la nuova colonna target dal resto del nuovo dataset.

Errori di Selezione in un menu a tendina (o simili)

Questa operazione di sporcatura consiste nel sostituire un valore categorico con un altro valore categorico valido nella feature relativa.

Questo tipo di errore può verificare se si utilizza un database in cui le variabili categoriche (e quindi anche quella target) si possono inserire solo selezionandole attraverso un menu' a tendina (o simile) nella cella relativa all' informazione da inserire.

Non tutte le features contengono variabili categoriche binarie, di conseguenza sono state eseguite due tipologie di soluzioni : swap e sostituzione randomica.

Di seguito l'implementazione con eventuali commenti dove necessario:

```
10  # Errori di selezione
11  #ipotizza che la compilazione di un dataset abbia alcuni elementi (categorici + target) compilabili attraverso la selezione di varie opzioni in un menu a tendina
12  #person_gender: ['female' 'male'] -> swap
13  #person_education: ['Master' 'High School' 'Bachelor' 'Associate' 'Doctorate'] -> random substitution
14  #person_home_ownership: ['RENT' 'OWN' 'MORTGAGE' 'OTHER'] -> random substitution
15  #loan_intent: ['PERSONAL' 'EDUCATION' 'MEDICAL' 'VENTURE' 'HOMEIMPROVEMENT' 'DEBTCONSOLIDATION'] -> random substitution
16  #loan_status (target): [0,1] -> swap
17
18  if selezioni > 0:
19
20      selectable_cols_name = ['person_gender', 'person_education', 'person_home_ownership', 'loan_intent', 'loan_status']
21      df_unified_subset = df_unified[selectable_cols_name].copy()
22
23      num_selection_errors = int(df_unified_subset.size * selezioni)
24      if num_selection_errors > 0:
25          # Ensure we don't try to add more selection errors than there are cells
26          num_selection_errors = min(num_selection_errors, df_unified_subset.size)
27          # Use random.sample for efficiency if adding many error values
28          all_cell_indices = [(r, c) for r in range(df_unified_subset.shape[0]) for c in range(df_unified_subset.shape[1])]
29          missing_cell_indices = random.sample(all_cell_indices, num_selection_errors)
30
31          person_gender = 0
32          person_education = 0
33          person_home_ownership = 0
34          loan_intent = 0
35          loan_status = 0
36
```

In questa sezione di codice viene fatto un sotto-dataset contenente solo le colonne categoriche interessate e fatti vari controlli sull' input.

Inoltre, vengono sia identificate le celle su cui operare che inizializzate le variabili utili per verificare la distribuzione delle modifiche effettuate rispetto alle features identificate.

```
38  for row_idx, col_idx in missing_cell_indices:
39      col_name = df_unified_subset.columns[col_idx] # Get column name from index
40
41      #potrebbe essere migliorata con una lista mappata, ma per ora lasciamo così (principalmente per testing,debugging ed affini)
42
43      if col_name == 'loan_status':
44          loan_status += 1
45          current_value = df_unified_subset.iloc[row_idx, col_idx]
46          new_value = 1 - current_value # Invert the value 1-1 = 0 ; 1-0 = 1
47          df_unified_subset.iloc[row_idx, col_idx] = new_value
48
49      elif col_name == 'person_gender':
50          person_gender += 1
51          current_value = df_unified_subset.iloc[row_idx, col_idx]
52          new_value = 'male' if current_value == 'female' else ('female' if current_value == 'male' else current_value)
53          df_unified_subset.iloc[row_idx, col_idx] = new_value
54
55      elif col_name == 'person_education':
56          person_education += 1
57          current_value = df_unified_subset.iloc[row_idx, col_idx]
58          list_of_values = ['Master', 'High School', 'Bachelor', 'Associate', 'Doctorate']
59          list_of_values.remove(current_value)
60          new_value = random.choice(list_of_values)
61          df_unified_subset.iloc[row_idx, col_idx] = new_value
62
```

È bene notare che è stato deciso di sporcare anche il target ('loan_status'), data la natura dell'errore.

```
70
71     elif col_name == 'person_home_ownership':
72         person_home_ownership += 1
73         current_value = df_unified_subset.iloc[row_idx, col_idx]
74         list_of_values = ['RENT', 'OWN', 'MORTGAGE', 'OTHER']
75         list_of_values.remove(current_value)
76         new_value = random.choice(list_of_values)
77         df_unified_subset.iloc[row_idx, col_idx] = new_value
78
79     elif col_name == 'loan_intent':
80         loan_intent += 1
81         current_value = df_unified_subset.iloc[row_idx, col_idx]
82         list_of_values = ['PERSONAL', 'EDUCATION', 'MEDICAL', 'VENTURE', 'HOMEIMPROVEMENT', 'DEBTCONSOLIDATION']
83         list_of_values.remove(current_value)
84         new_value = random.choice(list_of_values)
85         df_unified_subset.iloc[row_idx, col_idx] = new_value
86
87     else:
88         print(f"valore non gestito:{df_unified_subset.iloc[row_idx, col_idx]} , colonna: {col_name}, tipo: {type(df_unified_subset.iloc[row_idx, col_idx])} ")
```

All'interno di questo ciclo for per ogni cella da modificare viene identificata l'appartenenza di essa ad una feature ed, in base a ciò, vengono effettuate le operazioni di swap o sostituzione randomica.

Naturalmente, utilizzando una lista mappata l'algoritmo utilizzato risulterebbe il più efficiente, però al fine di una migliore leggibilità del programma e considerando la natura progettistica di esso, è stato scelto di utilizzare una cascata di "if-elif-else" per rendere più facile la lettura.

Nell'ultima riga (riga 88) è presente un comando di "print" al fine di debug che, se il programma lavora come inteso, non verrà eseguito.

```
81
82     ## aggiornamento del dataset
83     df_unified_dropped = df_unified.drop(columns=selectable_cols_name)
84     data_uncleaned = pd.concat([df_unified_dropped, df_unified_subset], axis=1)
85
86     ## divisione target e test
87     xv_train = data_uncleaned.drop('loan_status', axis=1) # loan status è il nome della target nel nostro dataset
88     yv_train = data_uncleaned['loan_status']
89
90     print('')
91     print('modifiche in:')
92     print(f'person_gender: {person_gender}')
93     print(f'person_education: {person_education}')
94     print(f'person_home_ownership: {person_home_ownership}')
95     print(f'loan_intent: {loan_intent}')
96     print(f'loan_status: {loan_status}')
97     print('')
```

In seguito al ciclo for vengono fatte le operazioni finali di aggiornamento del dataset e divisione della colonna target dal resto del dataset.

Inoltre, vengono stampate le informazioni utili per verificare la distribuzione delle sporcature nelle features categoriche.

Errori di input manuali

Questa operazione di sporcatura consiste nel simulare dei possibili errori di battitura durante l'inserimento manuale del dato nel database.

I tipi di errori di battitura implementati sono:

- Inserimento valore a singolo carattere errato specifico per il loan_status (vado nel dettaglio successivamente) (1 può diventare 2 oppure NaN; 0 può diventare 9 oppure NaN)
- Sostituzione di “.” Con “,” per i valori float
- Invertire la posizione dei primi due caratteri consecutivi per valori interi maggiori di 9 (es: 690 diventa 960)
- Nelle variabili string invertire l'ordine di due caratteri consecutivi nella stringa (es: ciao diventa caio) (Swap)
- Nelle variabili string duplicare un carattere randomico all'interno della stringa (es: ciao diventa ciao) (Duplicate)
- Nelle variabili string rendere il primo carattere della stringa maiuscolo se minuscolo e viceversa (es: ciao diventa Ciao e Carlo diventa carlo) (Case_change)
- Nelle variabili string aggiungere a fine stringa un numero randomico di caratteri “ ” (es: “ciao” diventa “ciao ”) (Spazio_multiplo)
- Inserire nelle variabili string dei caratteri speciali che il dataset potrebbe identificare come separatore (nel nostro caso “;” e tabulazione) (es: ciao diventa ci;ao) (Separatore)
- Inserire alla fine variabili string il carattere speciale che identifica l'andare a capo (“\n”) (es: ciao diventa ciao\n) (acapo)

Di seguito l'implementazione con eventuali commenti dove necessario:

```
# errore di input
# es: per un numerico 96 diventa 69, per il categorico ciao, ciao e simili, errori di maluscole-minuscole, spazi/ spazi multipli, caratteri speciali quali quello di acapo ";"
# in teoria posso sporcare anche il target senza incappare in errori di training
if input_err > 0:
    num_input_errors = int(df_unified.size * input_err)
    loan_status = 0
    float_point_to_comma = 0
    int_swap_first_two_digits = 0
    str_swap_consecutive_char=0
    str_duplicate_random_char=0
    str_first_case_change=0
    str_endswith_multiple_spaces=0
    str_add_random_separator=0
    str_add_new_line_char=0

    if num_input_errors > 0:
        # Ensure we don't try to add more input errors than there are cells
        num_input_errors = min(num_input_errors, df_unified.size)
        # Use random.sample for efficiency if adding many
        all_cell_indices = [(r, c) for r in range(df_unified.shape[0]) for c in range(df_unified.shape[1])]
        input_cell_indices = random.sample(all_cell_indices, num_input_errors)
```

Qui vengono inizializzate le variabili utili per verificare la distribuzione delle modifiche effettuate rispetto alle features identificate e fatti controlli preliminari sull' input.

```

171 for row_idx, col_idx in input_cell_indices:
172     col_name = df_unified.columns[col_idx] # Get column name from index
173     if col_name == 'loan_status':
174         loan_status +=1
175         df_unified.iloc[row_idx, col_idx] = random.choice([np.nan,2]) if (df_unified.iloc[row_idx, col_idx]) == 1 else random.choice([np.nan,9])
176
177 elif isinstance(df_unified.iloc[row_idx, col_idx], (float, np.floating)): # se è un numero non target float
178     value= df_unified.iloc[row_idx, col_idx]
179     df_unified.iloc[row_idx, col_idx] = f"{value:.2f}".replace('.', ',') # se è float sostituisci i . con le ,
180     float_point_to_comma +=1
181
182 elif isinstance(df_unified.iloc[row_idx, col_idx], (int, np.integer)): # se è un numero non target int:
183     value= df_unified.iloc[row_idx, col_idx]
184     if int(value) > 9:
185         number = int(df_unified.iloc[row_idx, col_idx])
186         str_number = str(number)
187         # Swap the first two digits
188         new_number = int(str_number[1] + str_number[0] + str_number[2:]) # swap dei primi due caratteri di un numero intero
189         df_unified.iloc[row_idx, col_idx] = new_number
190         int_swap_first_two_digits +=1

```

Data la natura della sporcatura del dataset che stiamo effettuando abbiamo ipotizzato che pure il target può essere soggetto ad errore di inserimento. Data la natura delicata della feature abbiamo dovuto trattare il suo errore di inserimento in maniera specifica.

Per fare le operazioni sui numeri prima sono state convertite in stringa e poi riconvertito in numero. Notare che nell'operazione sui valori float il valore non viene riconvertito. Questo è causato dal fatto che Python non riesce a convertire in float i valori con la virgola. Questo, come si vedrà in seguito, non causa problemi in quanto gli algoritmi di Machine learning stessi riescono a gestire la cosa.

```

202 elif isinstance(df_unified.iloc[row_idx, col_idx], str):
203     old_str = df_unified.iloc[row_idx, col_idx]
204     error_type = random.choice(['swap','duplicate','case_change','spazio_multiplo','separatore','acapo'])
205     if error_type == 'swap': # inverte l'ordine di due caratteri consecutivi
206         str_swap_consecutive_char +=1
207         if len(old_str) >= 2:
208             swap_index = random.randint(0, len(old_str) - 2)
209             new_str = list(old_str)
210             new_str[swap_index], new_str[swap_index + 1] = new_str[swap_index + 1], new_str[swap_index]
211             df_unified.iloc[row_idx, col_idx] = "".join(new_str)
212     elif error_type == 'duplicate': #duplica un carattere randomico
213         str_duplicate_random_char +=1
214         duplicate_index = random.randint(0, len(old_str) - 1)
215         new_str = list(old_str)
216         new_str.insert(duplicate_index, new_str[duplicate_index])
217         df_unified.iloc[row_idx, col_idx] = "".join(new_str)
218     elif error_type == 'case_change': #inverte la prima carattere in maiuscola o minuscola
219         str_first_case_change +=1
220         new_str = list(old_str)
221         new_str[0].swapcase()
222         df_unified.iloc[row_idx, col_idx] = "".join(new_str)
223     elif error_type == 'spazio_multiplo':
224         str_endswith_multiple_spaces +=1
225         new_str = old_str + ' ' * random.randint(1,5)
226         df_unified.iloc[row_idx, col_idx] = new_str
227     elif error_type == 'separatore':
228         str_add_random_separator +=1
229         new_str = old_str+ random.choice([';',' ']) # separatori sono ; e tabulazione(rari dataset)
230         df_unified.iloc[row_idx, col_idx] = new_str
231     elif error_type == 'acapo':
232         str_add_new_line_char +=1
233         new_str = old_str + '\n'
234         df_unified.iloc[row_idx, col_idx] = new_str
235     else:
236         print(f"valore non gestito:{df_unified.iloc[row_idx, col_idx]} , colonna: {col_name}, tipo: {type(df_unified.iloc[row_idx, col_idx])} ")

```

Nell' immagine sopra sono presenti tutte le sporcature per le variabili categoriche.

Alla fine è presente un else contente un print di debug. Se ci sono stati elementi non gestiti , verrà notificato.

```
237     print('')
238     print('input error types added:')
239     print(f'loan_status: {loan_status}')
240     print(f'float_point_to_comma: {float_point_to_comma}')
241     print(f'int_swap_first_two_digits: {int_swap_first_two_digits}')
242     print(f'str_swap_consecutive_char: {str_swap_consecutive_char}')
243     print(f'str_duplicate_random_char: {str_duplicate_random_char}')
244     print(f'str_first_case_change: {str_first_case_change}')
245     print(f'str_endswith_multiple_spaces: {str_endswith_multiple_spaces}')
246     print(f'str_add_random_separator: {str_add_random_separator}')
247     print(f'str_add_new_line_char: {str_add_new_line_char}')
248     print('')
249
```

Qui vengono stampate le informazioni utili per verificare la distribuzione delle sporcature nelle features del dataset.

Def fitEncoding(...)

Questa funzione viene chiamata solo dal def prepareData (...). Essa si occupa di fare l'encoding delle colonne categoriche in numeriche per permettere l'addestramento negli algoritmi che esplicitamente richiedono la cosa.

“fitEncoding” è definita con:

```
1 def fitEncoding(X_train,y_train,X_test,y_test):
```

inoltra presenta un output con:

```
33 |     return X_train_processed,yv_train,X_test_processed,yv_test
```

Ove gli output che presentano il decoratore “processed” sono il dataset che è stato modificato mentre gli altri no.

Di seguito l’implementazione con eventuali commenti dove necessario:

```
1 def fitEncoding(X_train,y_train,X_test,y_test):
2
3     xv_train = X_train.copy()
4     yv_train = y_train.copy()
5     xv_test = X_test.copy()
6     yv_test = y_test.copy()
7
8     categorical_cols = ['person_gender','person_education', 'person_home_ownership', 'loan_intent']
9
10    # fitting dell' encoder
11    encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False, drop='first')
12    encoder.fit(xv_train[categorical_cols])
13
14    encoded_features_train = encoder.transform(xv_train[categorical_cols])
15    encoded_features_test = encoder.transform(xv_test[categorical_cols])
16
17    # Crea DataFrame dalle feature codificate
18    # È cruciale usare gli stessi indici e nomi di colonna per una corretta unione
19    encoded_df_train = pd.DataFrame(encoded_features_train,
20                                     columns=encoder.get_feature_names_out(categorical_cols),
21                                     index=xv_train.index)
22
23    encoded_df_test = pd.DataFrame(encoded_features_test,
24                                    columns=encoder.get_feature_names_out(categorical_cols),
25                                    index=xv_test.index)
26
27    # Rimuovi le colonne categoriche originali e concatena le nuove colonne codificate
28    X_train_processed = pd.concat([xv_train.drop(columns=categorical_cols), encoded_df_train], axis=1)
29    X_test_processed = pd.concat([xv_test.drop(columns=categorical_cols), encoded_df_test], axis=1)
30
31
32
33    return X_train_processed,yv_train,X_test_processed,yv_test
```

Dopo aver identificato manualmente le colonne categoriche è stato creato l'encoder “encoder” con il OneHotEncoder che droppa la prima feature di quelle create così da evitare multicollinearità.

Dopo aver preliminarmente fissato l'encoder sulle categoriche di X_train e applicato l'encoder sia alle categoriche di X_train che X_test essi sono stati sistemati per usare gli stessi indici e nomi di colonna per una corretta unione.

Infine sono stati creati i nuovi "X_train_processed" ed "X_test_processed" concatenando le nuove categoriche nel vecchio dataset e droppando le categoriche vecchie.

Def evaluate_model(...)

Questa funzione serve per calcolare la accuracy, f1-score e AUC sul test e training, confusion matrix solo sul test.

"Evaluate_model" è definita come:

```
def evaluate_model(model, X_test, y_test, y_pred, x_train_type ,y_train ,y_train_pred ,y_pred_proba=None, y_train_pred_proba=None):
```

Di seguito il codice, commentato dove necessario:

```
3 # Calculate metrics test
4 accuracy = accuracy_score(y_test, y_pred)
5 f1 = f1_score(y_test, y_pred)
6 auc_test = roc_auc_score(y_test, y_pred_proba)
7 conf_matrix = confusion_matrix(y_test, y_pred)
8
9 # Calculate metrics Train (for overfitting)
10
11 acc_train = accuracy_score(y_train, y_train_pred)
12 f1_train = f1_score(y_train, y_train_pred)
13 auc_train = roc_auc_score(y_train, y_train_pred_proba)
14
15 #####
16
17 print(f"Model: {type(model).__name__}")
18 print(f"Accuracy test: {accuracy:.4f} | Accuracy Training: {acc_train:.4f}")
19 print(f"F1-score test: {f1:.4f} | F1-score Training: {f1_train:.4f}")
20 print(f"roc_AUC-score test:{auc_test:.4f} | roc_AUC-score Training: {auc_train:.4f}")
21
22 # Indice di Overfitting (differenza tra training e test performance)
23 # Una differenza maggiore indica più overfitting.
24 print(f"Differenza Accuracy: {acc_train - accuracy:.4f}")
25 print(f"Differenza F1-score: {f1_train - f1:.4f}")
26 print(f"Differenza AUC: {auc_train - auc_test:.4f}")
27 #####
```

```

28 # verifica features utilizzate
29 if type(model).__name__ != 'MLPClassifier':
30     feature_names = x_train_type.columns
31     # non è possibile stampare le features utilizzate per il neural a causa della mancanza nel dataframe
32     # utilizzato del comando per ottenere il nome delle colonne
33     # però i risultati del decision sono così simili che possiamo assumere che vengano utilizzate le stesse features o molto simili
34     if type(model).__name__ == 'DecisionTreeClassifier':
35         print("Importanza delle Feature per Decision Tree:")
36         # Ottieni l'importanza delle feature dall'attributo del modello
37         importances = model.feature_importances_
38         # Crea una serie per visualizzare meglio
39         feature_importances_dt = pd.Series(importances, index=feature_names).sort_values(ascending=False)
40
41         # Stampa l'importanza delle feature in formato testuale
42         for feature, importance in feature_importances_dt.items():
43             if importance > 0:
44                 if importance > 0.001:
45                     print(f'{feature}: {importance:.4f}')
46                 else:
47                     print(f'{feature}: {importance:.4e}')
48     elif type(model).__name__ == 'GaussianNB':
49
50         #stima dell' importanza
51         print("\nImportanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:")
52         # Calcola la permutation importance per Gaussian Naive Bayes
53         # Usiamo il test set per valutare l'importanza su dati non visti
54         perm_importance_gnb = permutation_importance(model, X_test, y_test, random_state=42)
55
56         # Crea una serie per visualizzare meglio, usando le medie dell'importanza
57         feature_importances_gnb = pd.Series(perm_importance_gnb.importances_mean, index=feature_names).sort_values(ascending=False)
58         for feature, importance in feature_importances_gnb.items():
59             if importance != 0 :
60                 if importance > 0.001 or  importance < -0.001 :
61                     print(f'{feature}: {importance:.4f}')
62                 else:
63                     print(f'{feature}: {importance:.4e}')
64
65     else: #test MLPClass
66         feature_names_MLP = x_train_type.columns
67         #print(list(feature_names_MLP))
68         print("\nImportanza delle Feature (stimata con Permutation Importance) per MLP:")
69         # Convert X_test to a NumPy array before calculating permutation importance
70         perm_importance_mlp = permutation_importance(model, X_test.values, y_test, random_state=42)
71
72         # Crea una serie per visualizzare meglio, usando le medie dell' importanza
73         feature_importances_mlp = pd.Series(perm_importance_mlp.importances_mean, index=feature_names_MLP).sort_values(ascending=False)
74
75         # Print the entire series to see all importance scores
76         print(feature_importances_mlp)
77
78
79     # Calculate ROC curve and AUC
80     if y_pred_proba is not None:
81         fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
82         roc_auc = auc(fpr, tpr)
83         print(f"AUC: {roc_auc:.4f}")
84
85         # Plot ROC curve
86         plt.figure(figsize=(8, 6))
87         plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
88         plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
89         plt.xlim([0.0, 1.0])
90         plt.ylim([0.0, 1.05])
91         plt.xlabel('False Positive Rate')
92         plt.ylabel('True Positive Rate')
93         plt.title('Receiver Operating Characteristic (ROC) Curve')
94         plt.legend(loc="lower right")
95         plt.show()
96     else:
97         print("AUC not calculated as predicted probabilities were not provided.")
98
99
100    # Plot Confusion Matrix
101    plt.figure(figsize=(8, 6))
102    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
103                xticklabels=['Predicted 0', 'Predicted 1'],
104                yticklabels=['Actual 0', 'Actual 1'])
105    plt.xlabel('Predicted Label')
106    plt.ylabel('True Label')
107    plt.title('Confusion Matrix')
108    plt.show()

```

Esecuzione test ed Analisi risultati

Caso pulito

Di seguito i valori ottenuti dall' esecuzione dei vari algoritmi di apprendimento senza l'inserimento di sporcature di alcun tipo.

Questi valori verranno utilizzati per i confronti futuri.

Gli elementi che andremo ad analizzare per i confronti sono:

- Accuracy test
- Accuracy Training
- F1-score test
- F1-score Training
- roc_AUC-score test
- roc_AUC-score Training
- Importanza delle features.

Avere i valori sia relativi al test che al training sono utili per notare eventuali overfitting od underfitting.

È bene notare come l'importanza delle features nel Gaussian Naive Bayes e nel MLP sono stimate con la “Permutation Importance” quindi non sono valori reali ma indicativi.

In particolare, spesso il MLPClassifier conterrà tutti i valori pari a 0 o valori negativi e quasi mai avrà la somma dell' importanza pari ad 1.

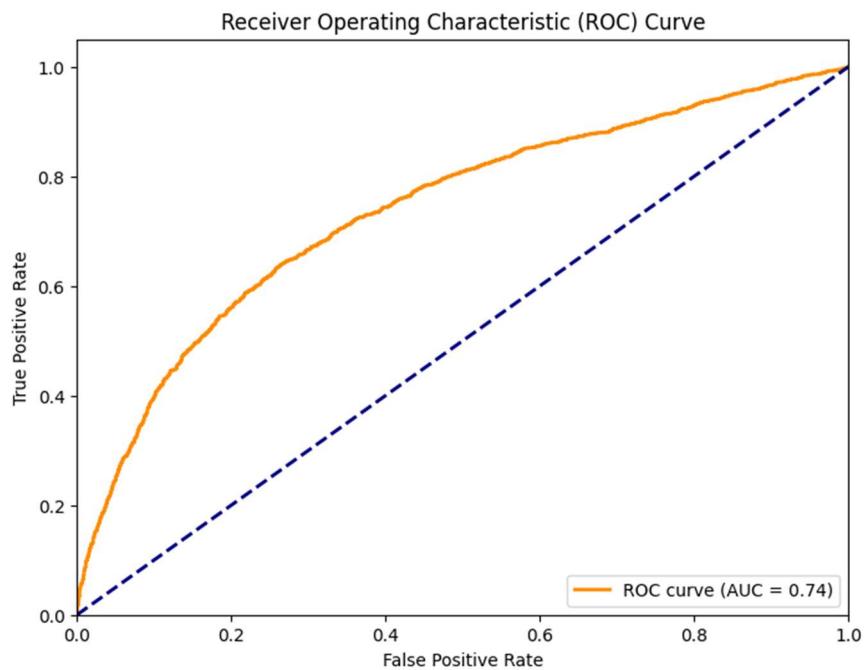
Questo è causato, per l'appunto, dalla “Permutation Importance” e suggerisce che permutando le features individualmente non diminuisce l'accuracy nel test set (anzi, se i valori sono negativi significa che le performance aumentano).

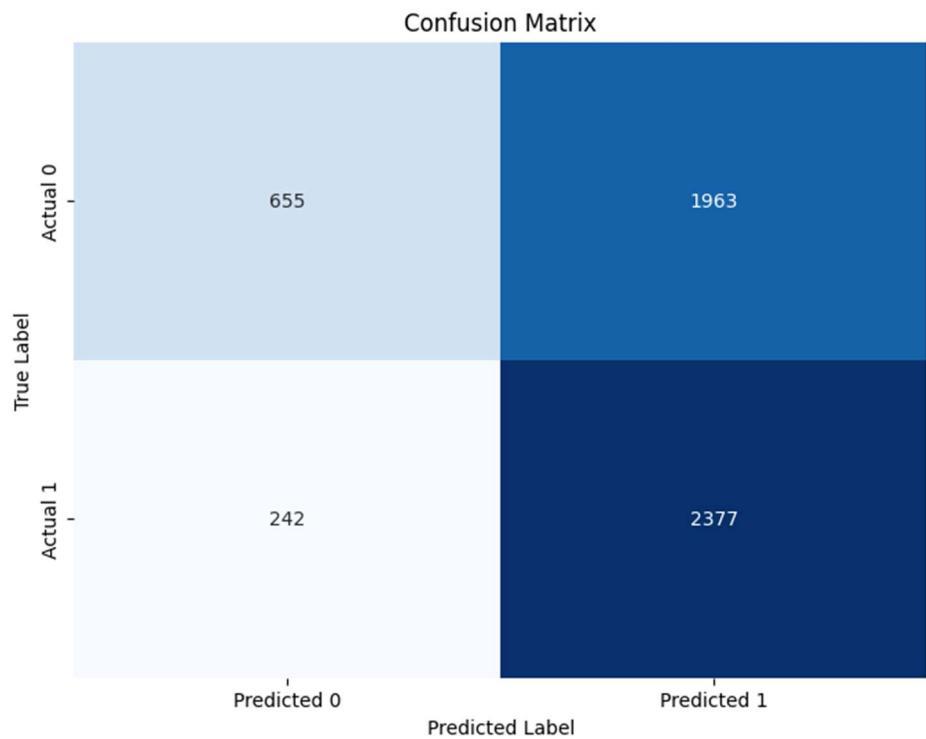
```

Model: GaussianNB
Accuracy test: 0.5790 | Accuracy Training: 0.5830
F1-score test: 0.6831 | F1-score Training: 0.6857
roc_auc-score test: 0.7371 | roc_auc-score Training: 0.7301
Differenza Accuracy: 0.0041
Differenza F1-score: 0.0026
Differenza AUC: -0.0069

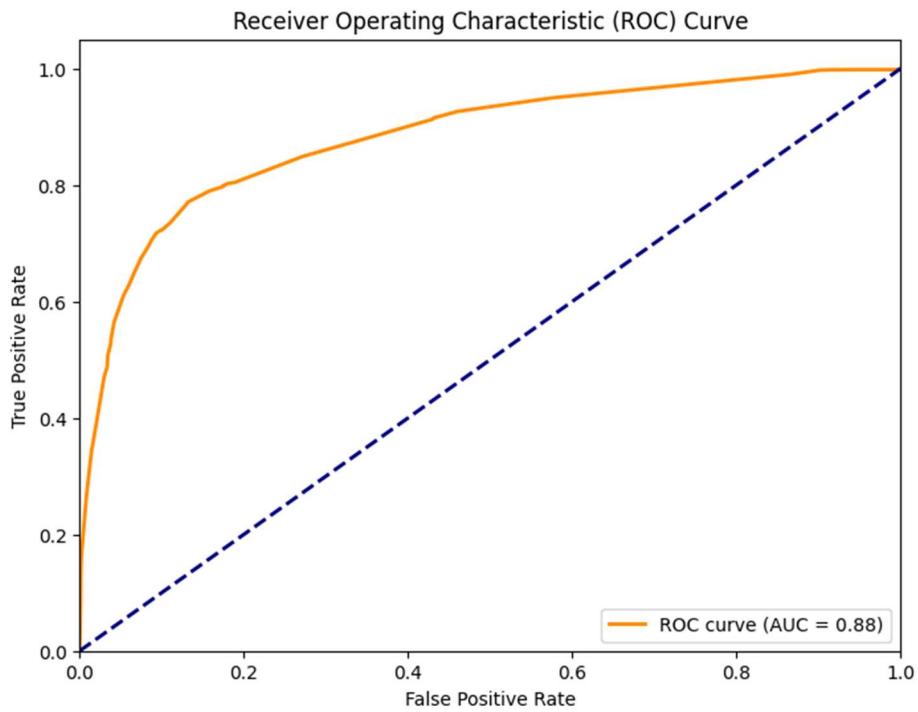
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'person_income': 0.0379
'loan_int_rate': 0.0230
'credit_score': 0.0077
'loan_amnt': 0.0026
'person_home_ownership_RENT': 7.6380e-04
'loan_intent_VENTURE': 1.9095e-04
'loan_intent_MEDICAL': 3.8190e-05
'loan_intent_PERSONAL': -7.6380e-05
'loan_intent_EDUCATION': -1.1457e-04
'loan_intent_HOMEIMPROVEMENT': -1.5276e-04
'person_education_High School': -1.9095e-04
'person_education_Bachelor': -1.9095e-04
'person_education_Master': -1.9095e-04
'cb_person_cred_hist_length': -0.0038
'person_age': -0.0066
'person_emp_exp': -0.0070
AUC: 0.7371

```

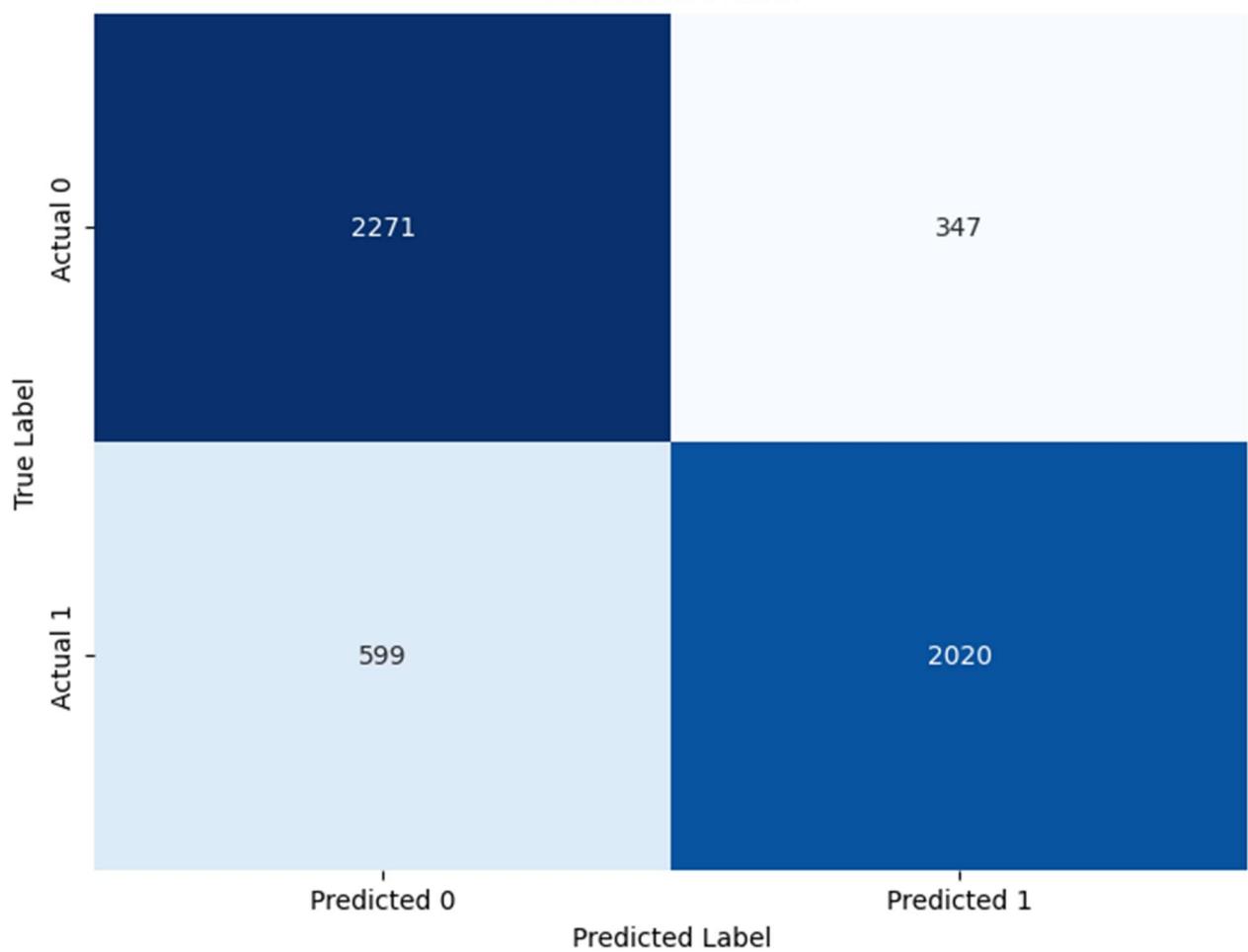




```
Model: DecisionTreeClassifier
Accuracy test: 0.8194 | Accuracy Training: 0.8243
F1-score test: 0.8103 | F1-score Training: 0.8141
roc_AUC-score test: 0.8820 | roc_AUC-score Training: 0.8923
Differenza Accuracy: 0.0049
Differenza F1-score: 0.0038
Differenza AUC: 0.0103
Importanza delle Feature per Decision Tree:
'person_income': 0.3149
'loan_int_rate': 0.2897
'loan_amnt': 0.1791
'person_home_ownership_RENT': 0.1333
'credit_score': 0.0300
'loan_intent_HOMEIMPROVEMENT': 0.0169
'person_home_ownership_OWN': 0.0141
'loan_intent_MEDICAL': 0.0114
'loan_intent_VENTURE': 0.0079
'person_age': 0.0018
'cb_person_cred_hist_length': 0.0011
AUC: 0.8820
```

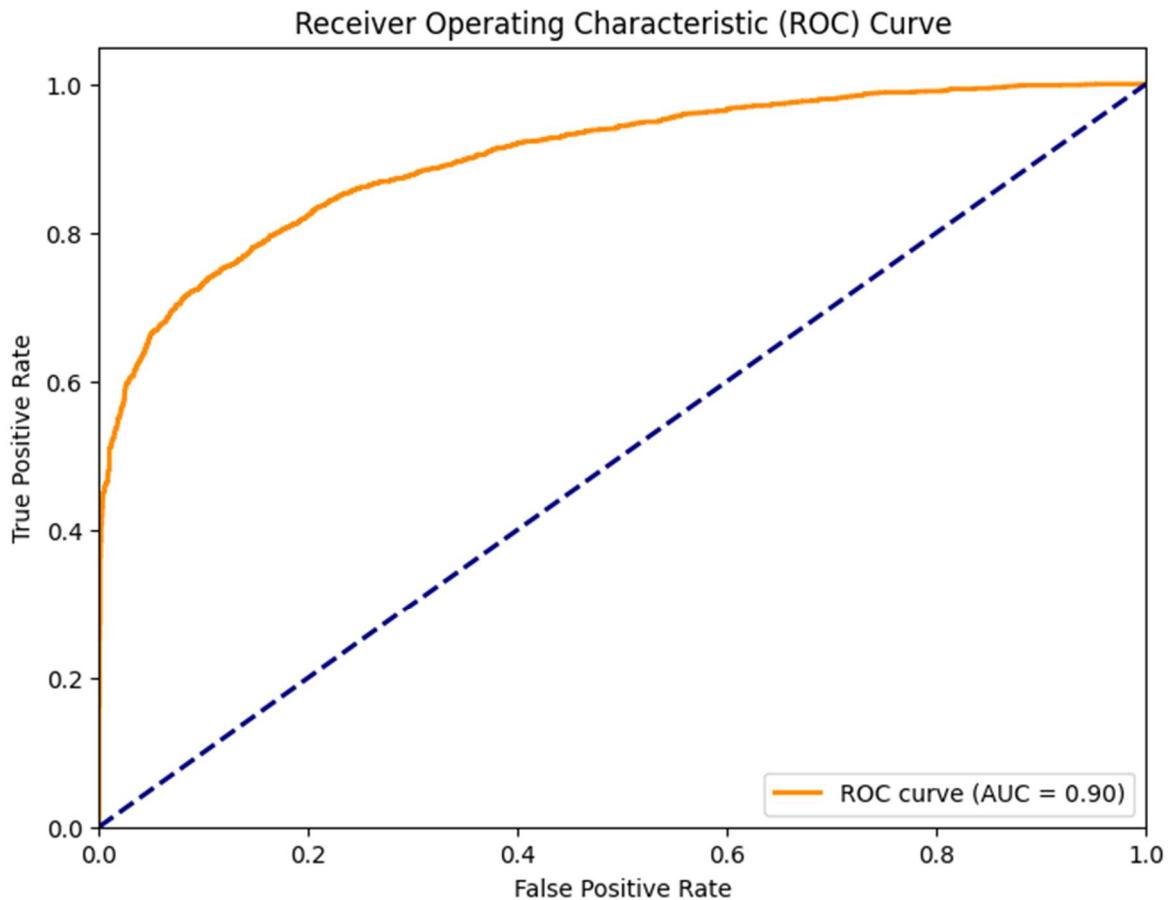


Confusion Matrix

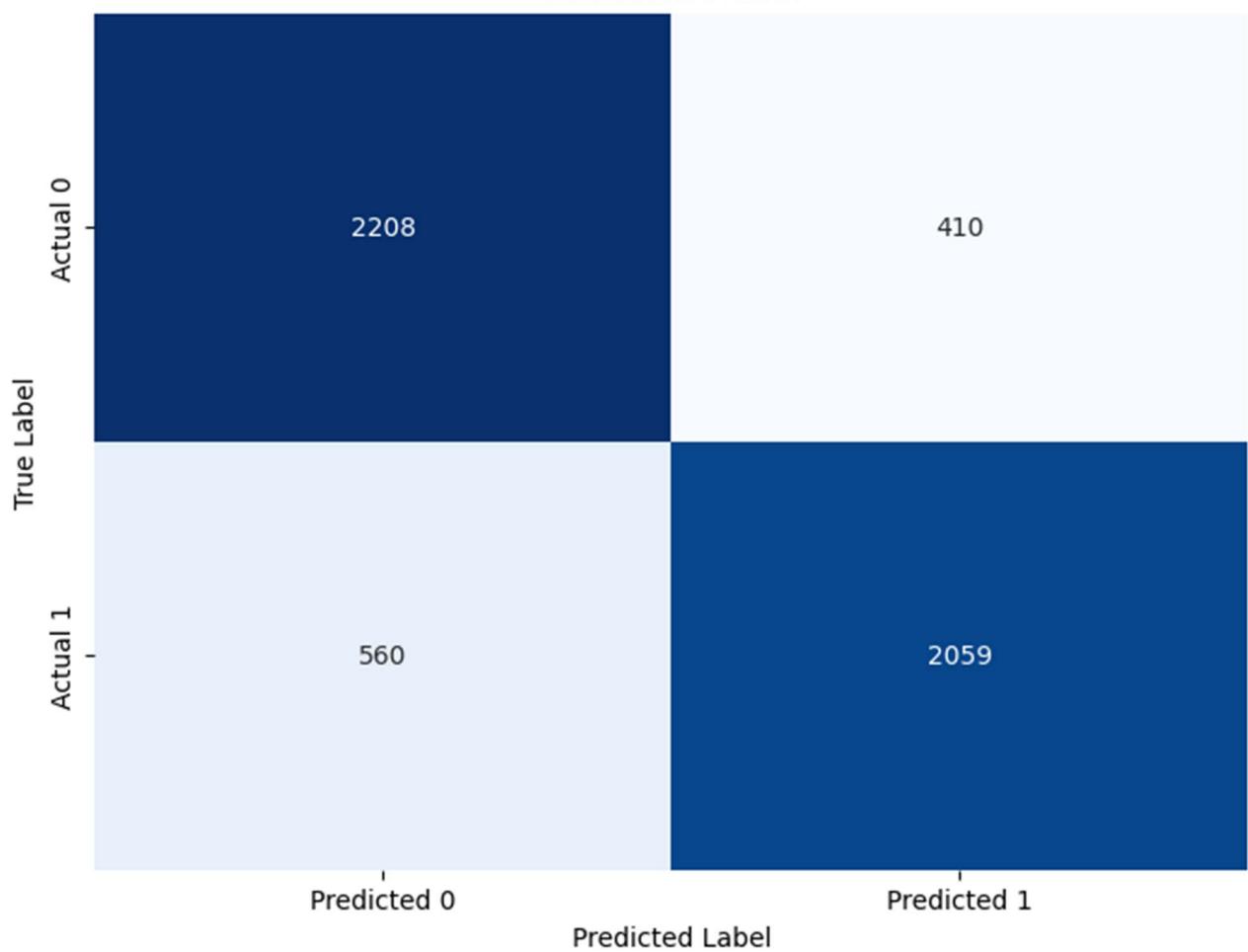


```
Model: MLPClassifier
Accuracy test: 0.8148 | Accuracy Training: 0.8509
F1-score test: 0.8094 | F1-score Training: 0.8455
roc_AUC-score test: 0.9008 | roc_AUC-score Training: 0.9276
Differenza Accuracy: 0.0361
Differenza F1-score: 0.0361
Differenza AUC: 0.0269

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age 0.0
person_income 0.0
person_emp_exp 0.0
loan_amnt 0.0
loan_int_rate 0.0
cb_person_cred_hist_length 0.0
credit_score 0.0
person_gender_male 0.0
person_education_Bachelor 0.0
person_education_Doctorate 0.0
person_education_High School 0.0
person_education_Master 0.0
person_home_ownership_OTHER 0.0
person_home_ownership_OWN 0.0
person_home_ownership_RENT 0.0
loan_intent_EDUCATION 0.0
loan_intent_HOMEIMPROVEMENT 0.0
loan_intent_MEDICAL 0.0
loan_intent_PERSONAL 0.0
loan_intent_VENTURE 0.0
dtype: float64
AUC: 0.9008
```



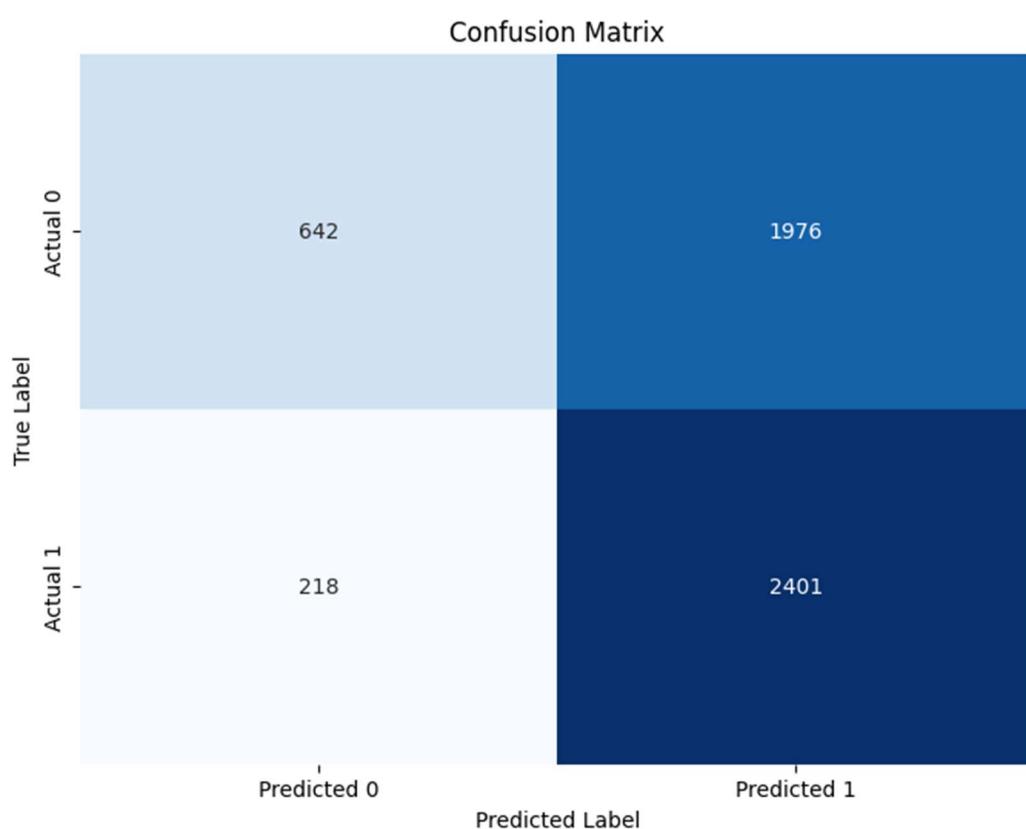
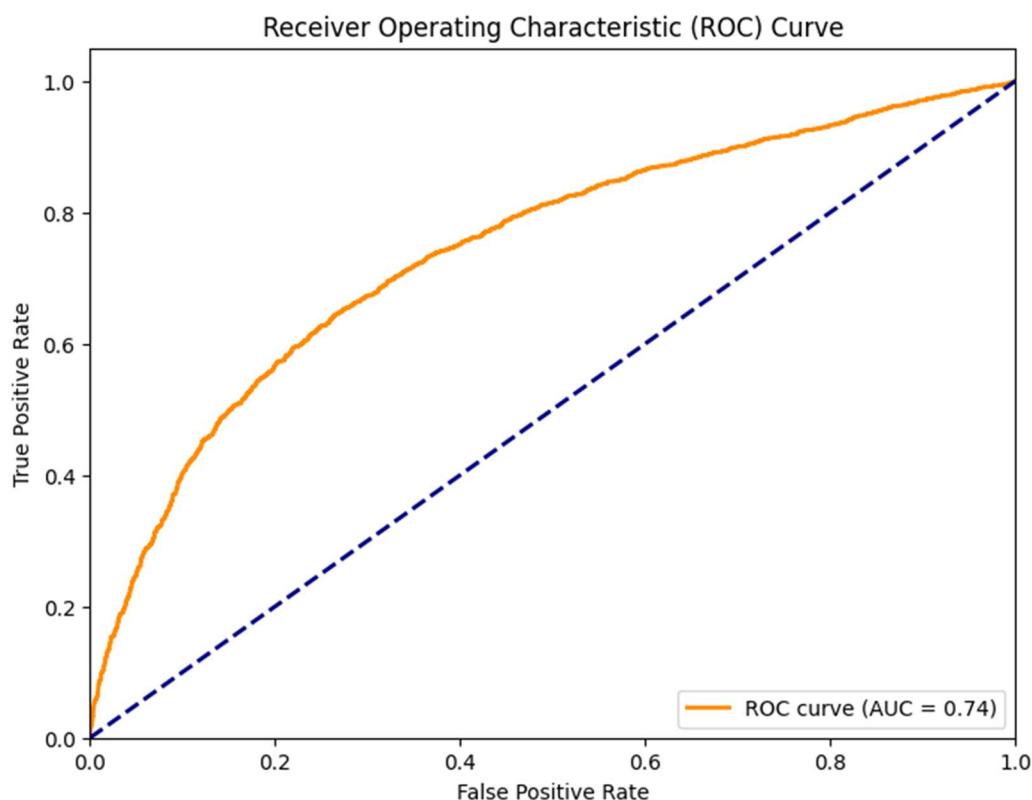
Confusion Matrix



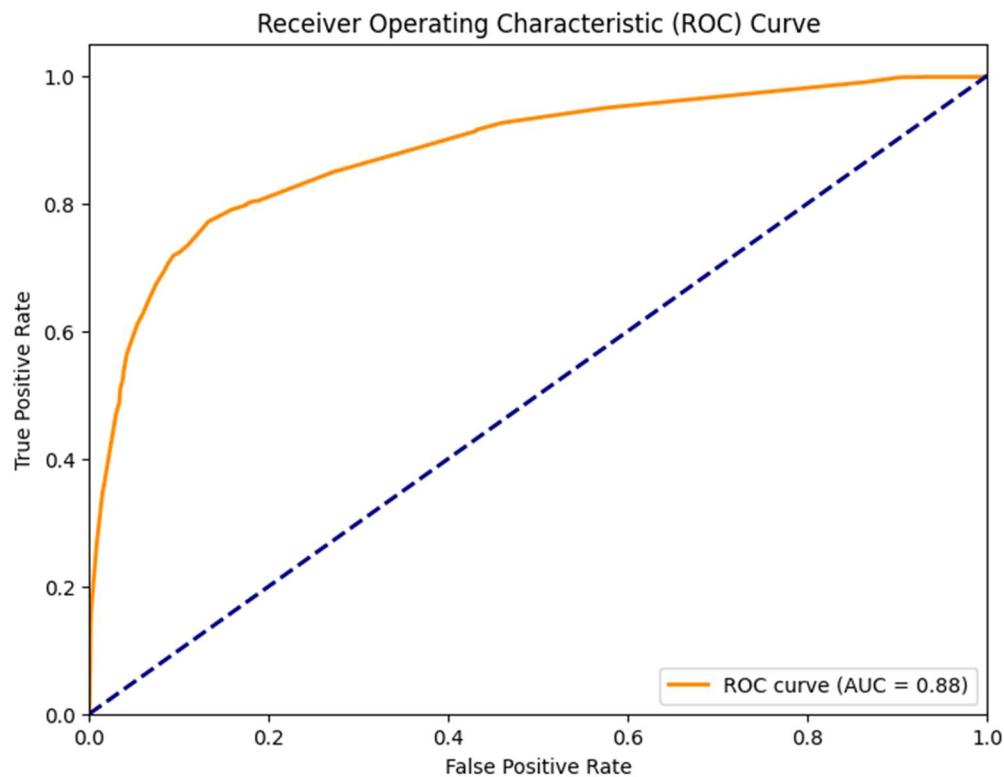
Inconsistenza dei dati 10%,90%

Inconsistenza nel 10% dei valori possibili nel training set:

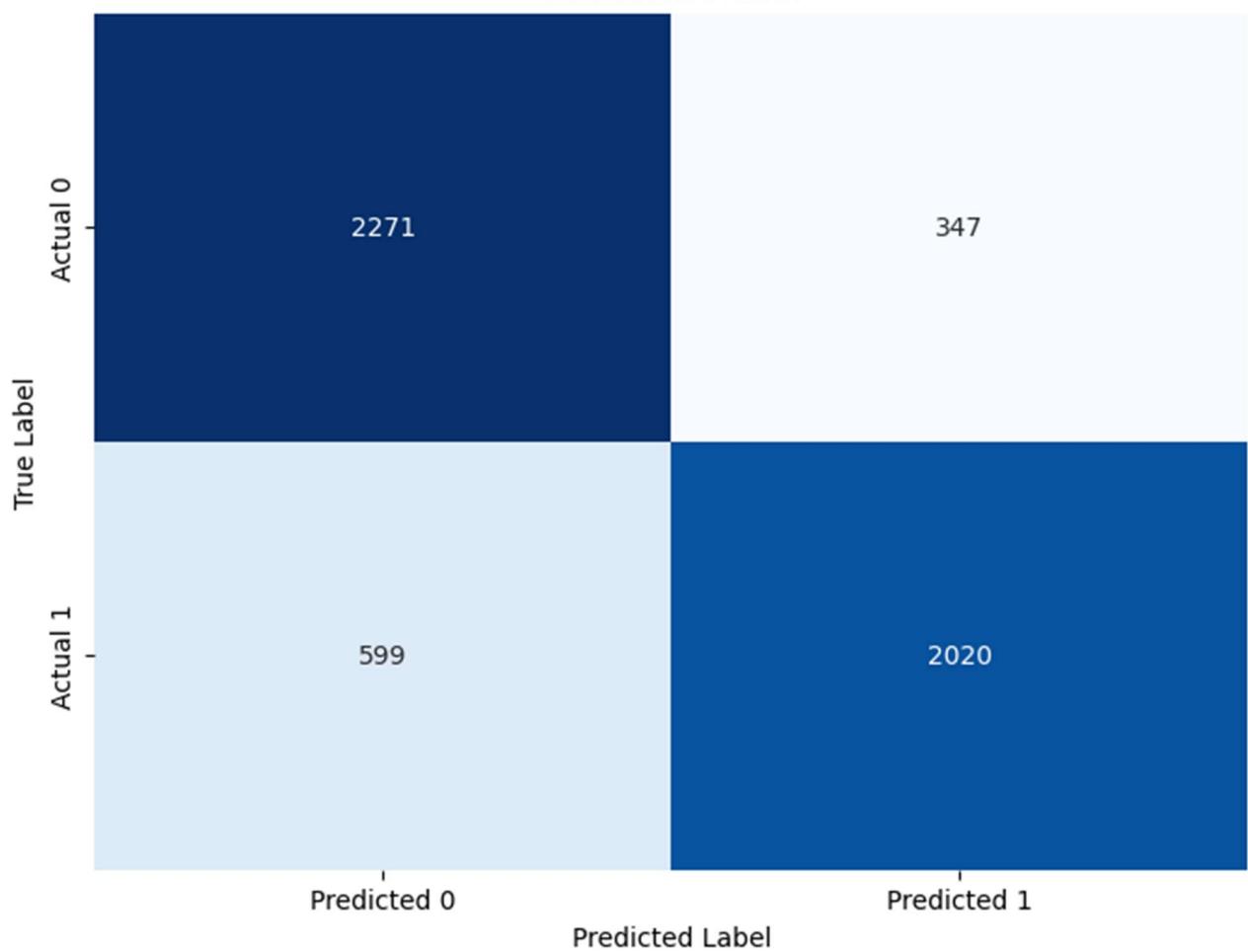
```
inconsistenze in:  
person_emp_exp: 1176  
cb_person_cred_hist_length: 1267  
  
Model: GaussianNB  
Accuracy test: 0.5811 | Accuracy Training: 0.5874  
F1-score test: 0.6864 | F1-score Training: 0.6884  
roc_AUC-score test: 0.7427 | roc_AUC-score Training: 0.7317  
Differenza Accuracy: 0.0064  
Differenza F1-score: 0.0020  
Differenza AUC: -0.0110  
  
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:  
'person_income': 0.0392  
'loan_int_rate': 0.0235  
'credit_score': 0.0071  
'loan_amnt': 0.0027  
'person_home_ownership_RENT': 6.1104e-04  
'person_home_ownership_own': 1.9095e-04  
'loan_intent_VENTURE': 3.8190e-05  
'loan_intent_PERSONAL': -3.8190e-05  
'person_education_High School': -3.8190e-05  
'person_education_Bachelor': -3.8190e-05  
'loan_intent_MEDICAL': -1.1457e-04  
'loan_intent_EDUCATION': -1.1457e-04  
'person_education_Master': -1.1457e-04  
'loan_intent_HOMEIMPROVEMENT': -1.5276e-04  
'cb_person_cred_hist_length': -4.5828e-04  
'person_emp_exp': -0.0041  
'person_age': -0.0063  
AUC: 0.7427
```



```
Model: DecisionTreeClassifier
Accuracy test: 0.8194 | Accuracy Training: 0.8243
F1-score test: 0.8103 | F1-score Training: 0.8141
roc_AUC-score test: 0.8820 | roc_AUC-score Training: 0.8923
Differenza Accuracy: 0.0049
Differenza F1-score: 0.0038
Differenza AUC: 0.0103
Importanza delle Feature per Decision Tree:
'person_income': 0.3149
'loan_int_rate': 0.2897
'loan_amnt': 0.1791
'person_home_ownership_RENT': 0.1333
'credit_score': 0.0300
'loan_intent_HOMEIMPROVEMENT': 0.0169
'person_home_ownership_OWN': 0.0141
'loan_intent_MEDICAL': 0.0114
'loan_intent_VENTURE': 0.0079
'person_age': 0.0018
'cb_person_cred_hist_length': 0.0011
AUC: 0.8820
```

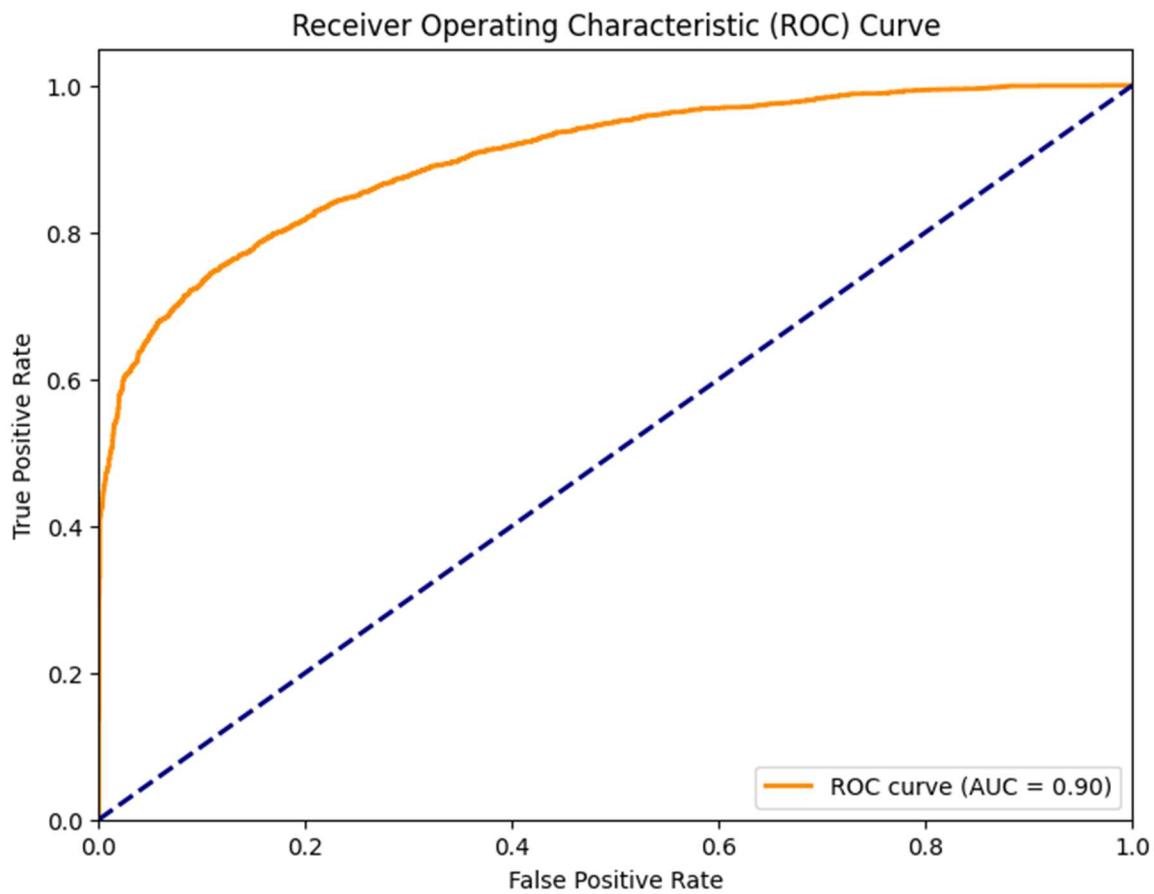


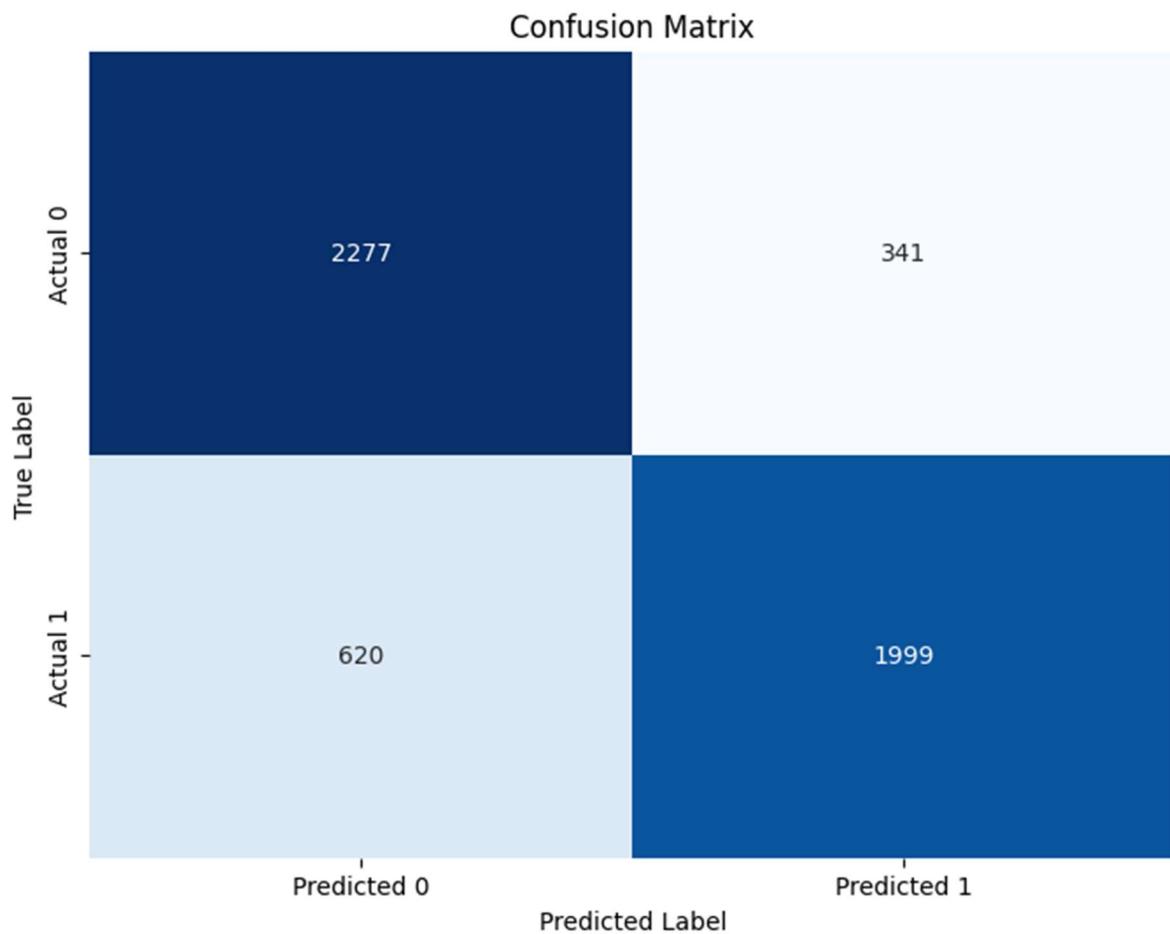
Confusion Matrix



```
Model: MLPClassifier
Accuracy test: 0.8165 | Accuracy Training: 0.8487
F1-score test: 0.8062 | F1-score Training: 0.8418
roc_AUC-score test: 0.9017 | roc_AUC-score Training:      0.9291
Differenza Accuracy: 0.0322
Differenza F1-score: 0.0356
Differenza AUC:      0.0274

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age                  0.0
person_income                 0.0
person_emp_exp                0.0
loan_amnt                     0.0
loan_int_rate                  0.0
cb_person_cred_hist_length    0.0
credit_score                   0.0
person_gender_male             0.0
person_education_Bachelor     0.0
person_education_Doctorate    0.0
person_education_High School   0.0
person_education_Master       0.0
person_home_ownership_OTHER    0.0
person_home_ownership_OWN      0.0
person_home_ownership_RENT     0.0
loan_intent_EDUCATION          0.0
loan_intent_HOMEIMPROVEMENT   0.0
loan_intent_MEDICAL            0.0
loan_intent_PERSONAL           0.0
loan_intent_VENTURE            0.0
dtype: float64
AUC: 0.9017
```





Ho in seguito testato valori crescenti di sporcatura aumentando di un decimo percentile il valore di inconsistenza ogni volta, ma le variazioni si sono rivelate minime.

Di seguito riporto solo i risultati al 90% del training set sporco ed infine farò dei commenti finali su di esso.

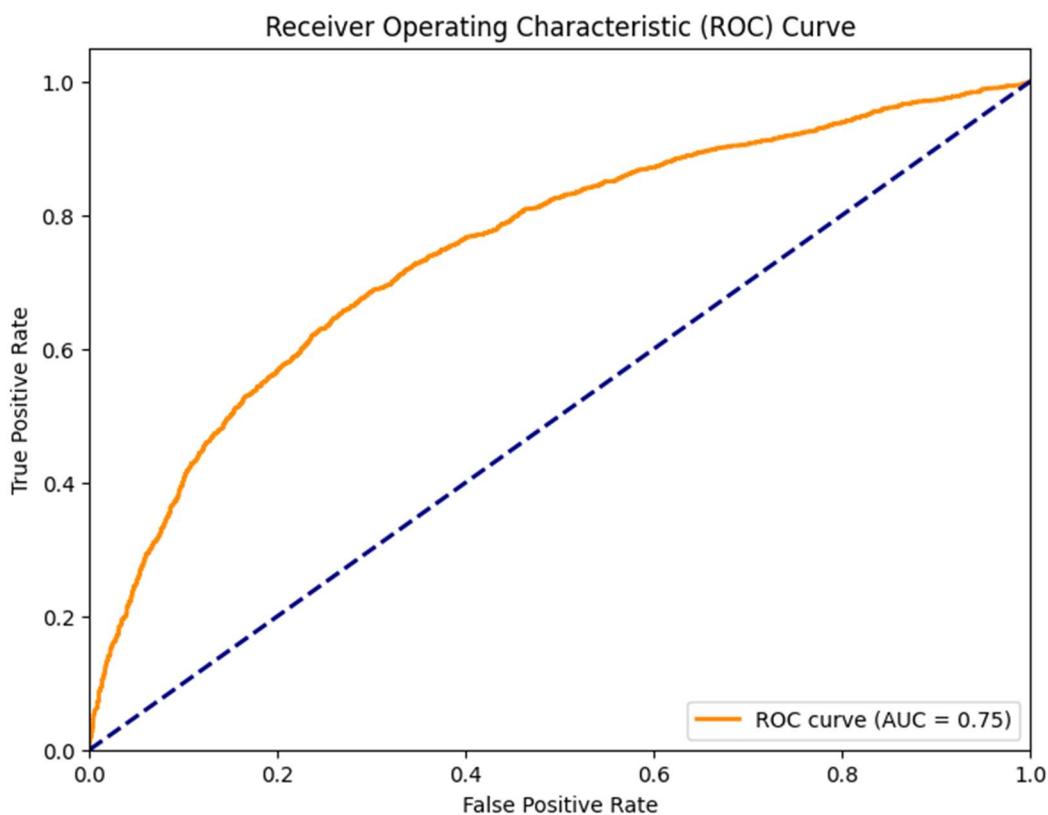
```

inconsistenze in:
person_emp_exp: 10971
cb_person_cred_hist_length: 11023

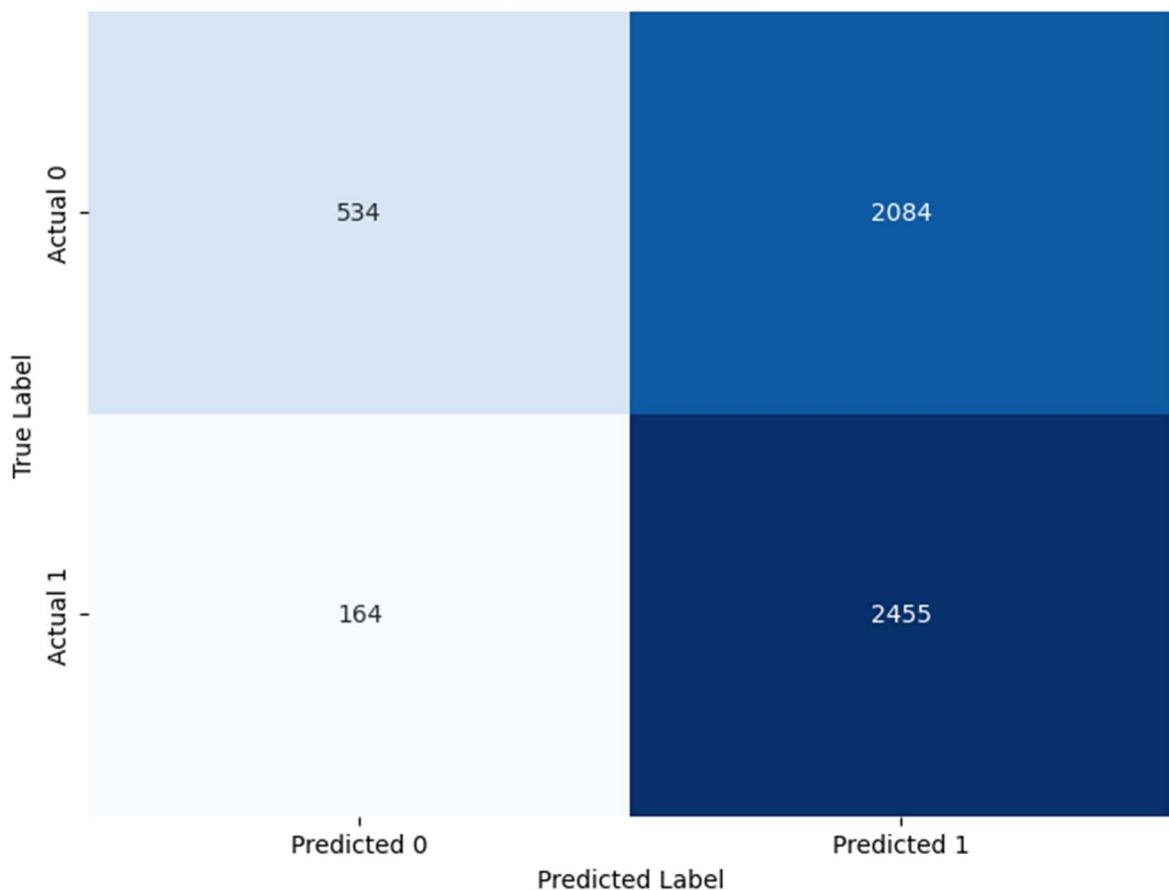
Model: GaussianNB
Accuracy test: 0.5707 | Accuracy Training: 0.5811
F1-score test: 0.6859 | F1-score Training: 0.6833
roc_AUC-score test: 0.7491 | roc_AUC-score Training: 0.7264
Differenza Accuracy: 0.0104
Differenza F1-score: -0.0026
Differenza AUC: -0.0228

Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'person_income': 0.0380
'loan_int_rate': 0.0195
'credit_score': 0.0067
'loan_amnt': 0.0018
'person_home_ownership_RENT': 0.0010
'loan_intent_VENTURE': 3.0552e-04
'loan_intent_MEDICAL': 1.9095e-04
'loan_intent_PERSONAL': 1.1457e-04
'person_education_High School': 1.1457e-04
'person_gender_male': 3.8190e-05
'person_education_Bachelor': -7.6380e-05
'loan_intent_HOMEIMPROVEMENT': -7.6380e-05
'person_home_ownership_OWN': -3.0552e-04
'loan_intent_EDUCATION': -4.5828e-04
'cb_person_cred_hist_length': -4.9647e-04
'person_emp_exp': -0.0031
'person_age': -0.0048
AUC: 0.7491

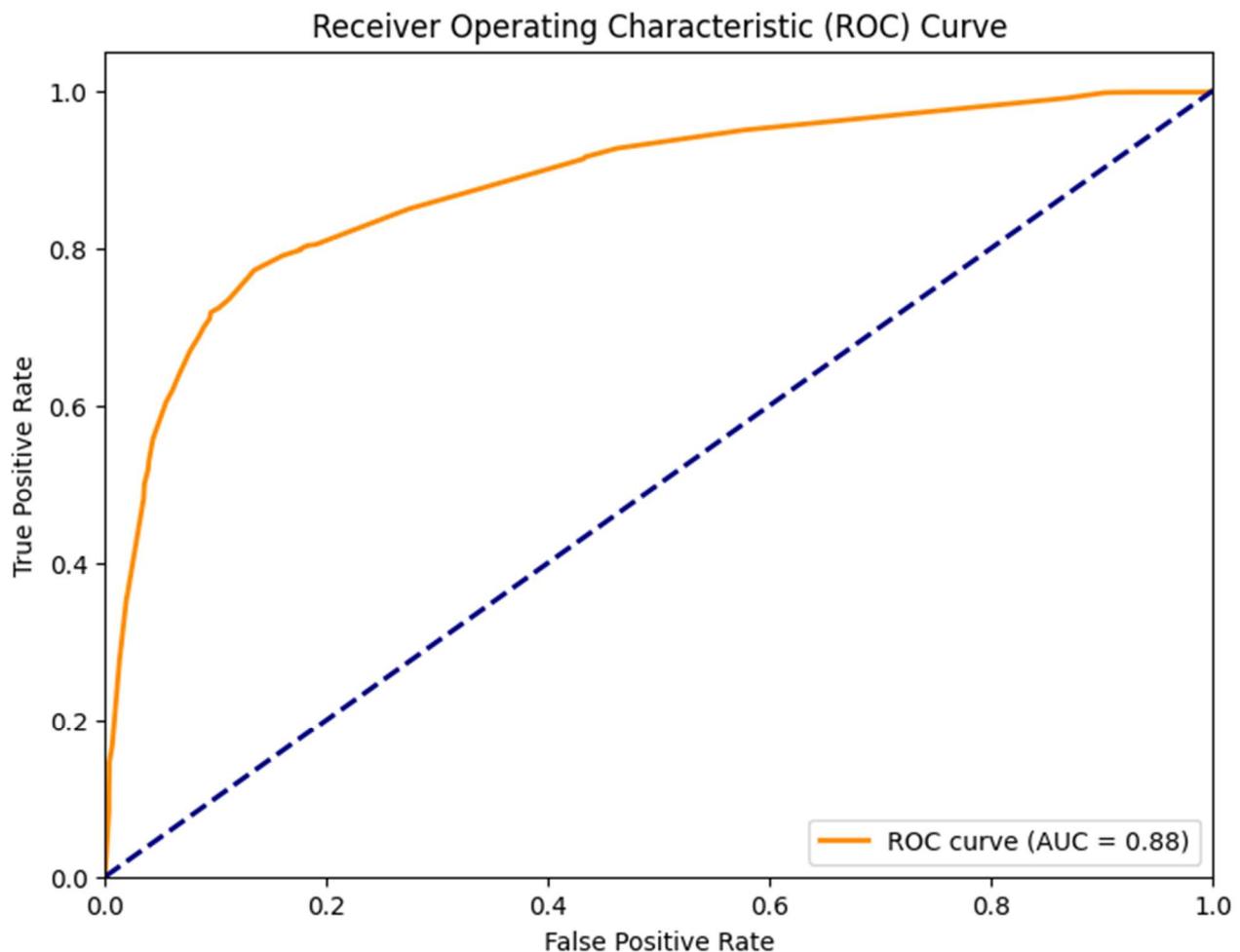
```



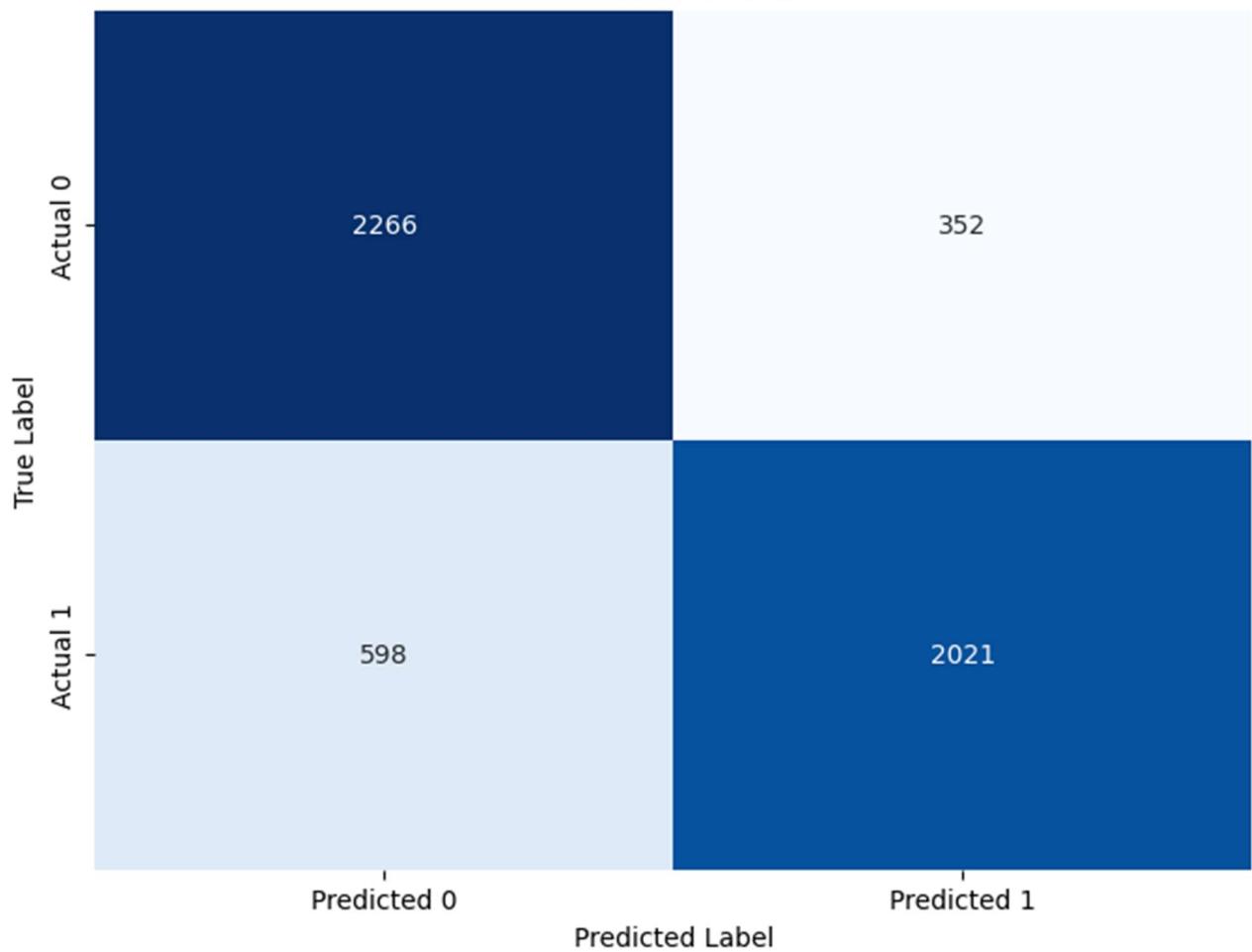
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.8186 | Accuracy Training: 0.8244
F1-score test: 0.8097 | F1-score Training: 0.8142
roc_AUC-score test: 0.8795 | roc_AUC-score Training: 0.8924
Differenza Accuracy: 0.0058
Differenza F1-score: 0.0045
Differenza AUC: 0.0129
Importanza delle Feature per Decision Tree:
'person_income': 0.3147
'loan_int_rate': 0.2893
'loan_amnt': 0.1789
'person_home_ownership_RENT': 0.1332
'credit_score': 0.0301
'loan_intent_HOMEIMPROVEMENT': 0.0168
'person_home_ownership_OWN': 0.0141
'loan_intent_MEDICAL': 0.0114
'loan_intent_VENTURE': 0.0079
'cb_person_cred_hist_length': 0.0019
'person_age': 0.0018
AUC: 0.8795
```



Confusion Matrix

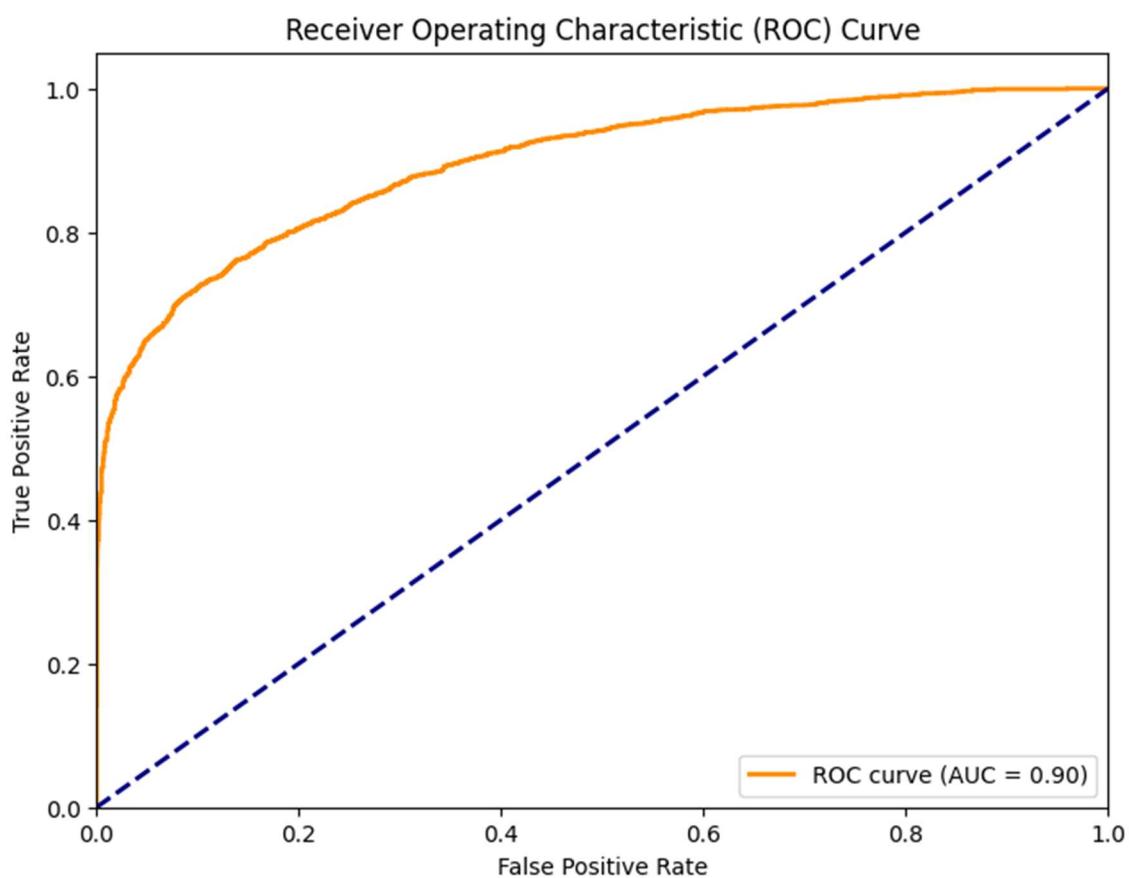


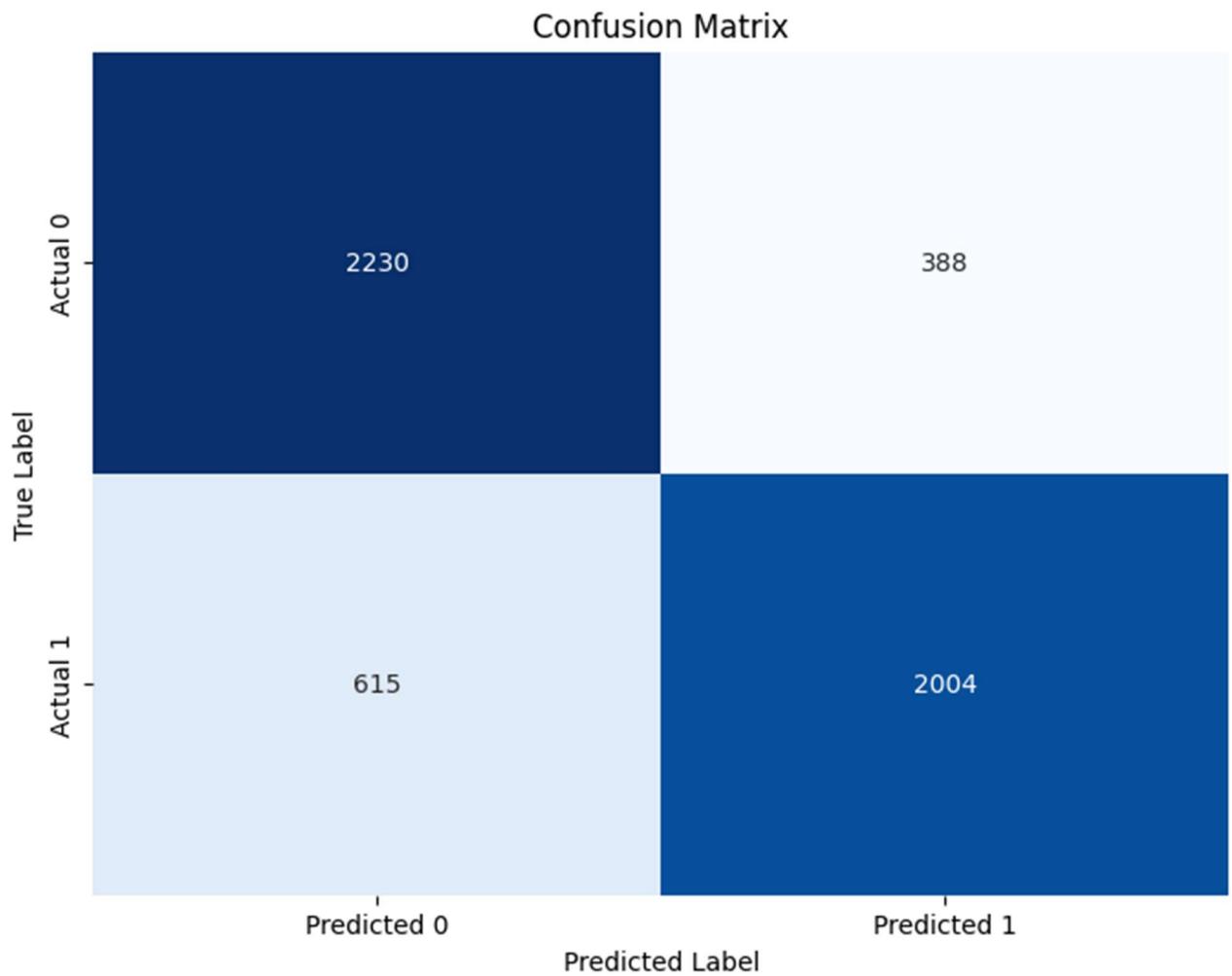
```

Model: MLPClassifier
Accuracy test: 0.8085 | Accuracy Training: 0.8508
F1-score test: 0.7998 | F1-score Training: 0.8426
roc_AUC-score test: 0.8958 | roc_AUC-score Training: 0.9301
Differenza Accuracy: 0.0423
Differenza F1-score: 0.0427
Differenza AUC: 0.0343

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age 0.0
person_income 0.0
person_emp_exp 0.0
loan_amnt 0.0
loan_int_rate 0.0
cb_person_cred_hist_length 0.0
credit_score 0.0
person_gender_male 0.0
person_education_Bachelor 0.0
person_education_Doctorate 0.0
person_education_High School 0.0
person_education_Master 0.0
person_home_ownership_OTHER 0.0
person_home_ownership_OWN 0.0
person_home_ownership_RENT 0.0
loan_intent_EDUCATION 0.0
loan_intent_HOMEIMPROVEMENT 0.0
loan_intent_MEDICAL 0.0
loan_intent_PERSONAL 0.0
loan_intent_VENTURE 0.0
dtype: float64
AUC: 0.8958

```





Notare come nell' albero decisionale e nell'MLP i valori sono pressoché identici sia nelle metriche numeriche che nelle rappresentazioni grafiche.

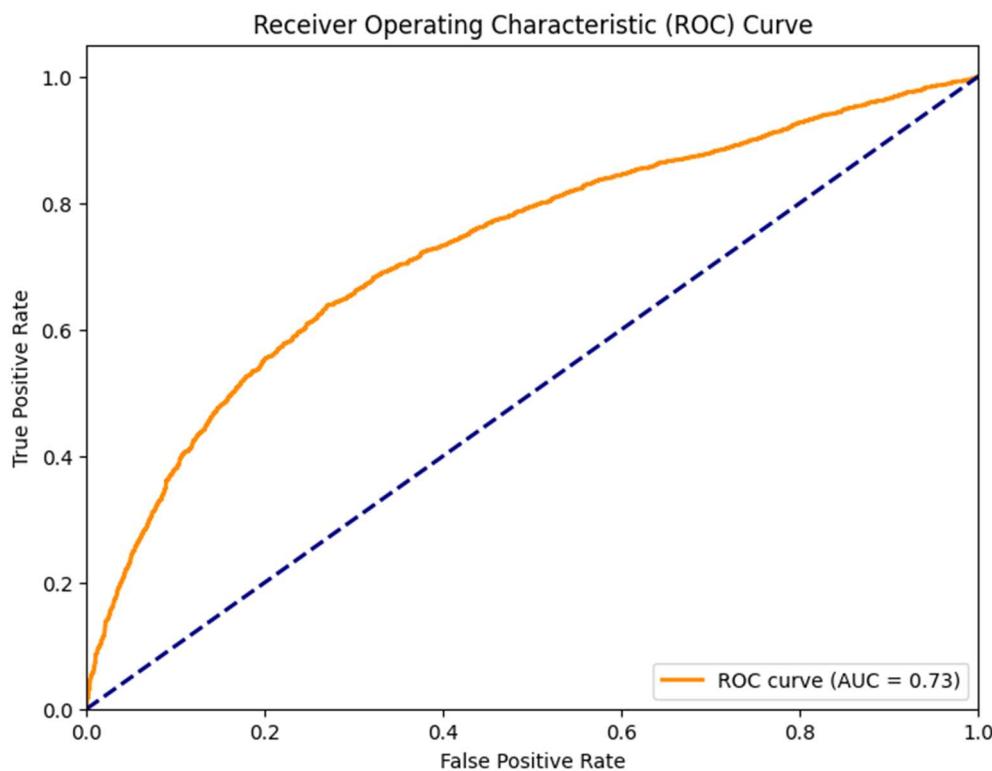
Per il Naive-Bayes sembra esserci un leggero miglioramento sull'AUC ma, attraverso la confusion matrix, scopriamo come questo miglioramento è dettato solo dall' aumento dei true-positive e non considera l'aumento dei false-positive.

Quindi pure il Naive-Bayes è relativamente solido rispetto a questo tipo di sporcatura.

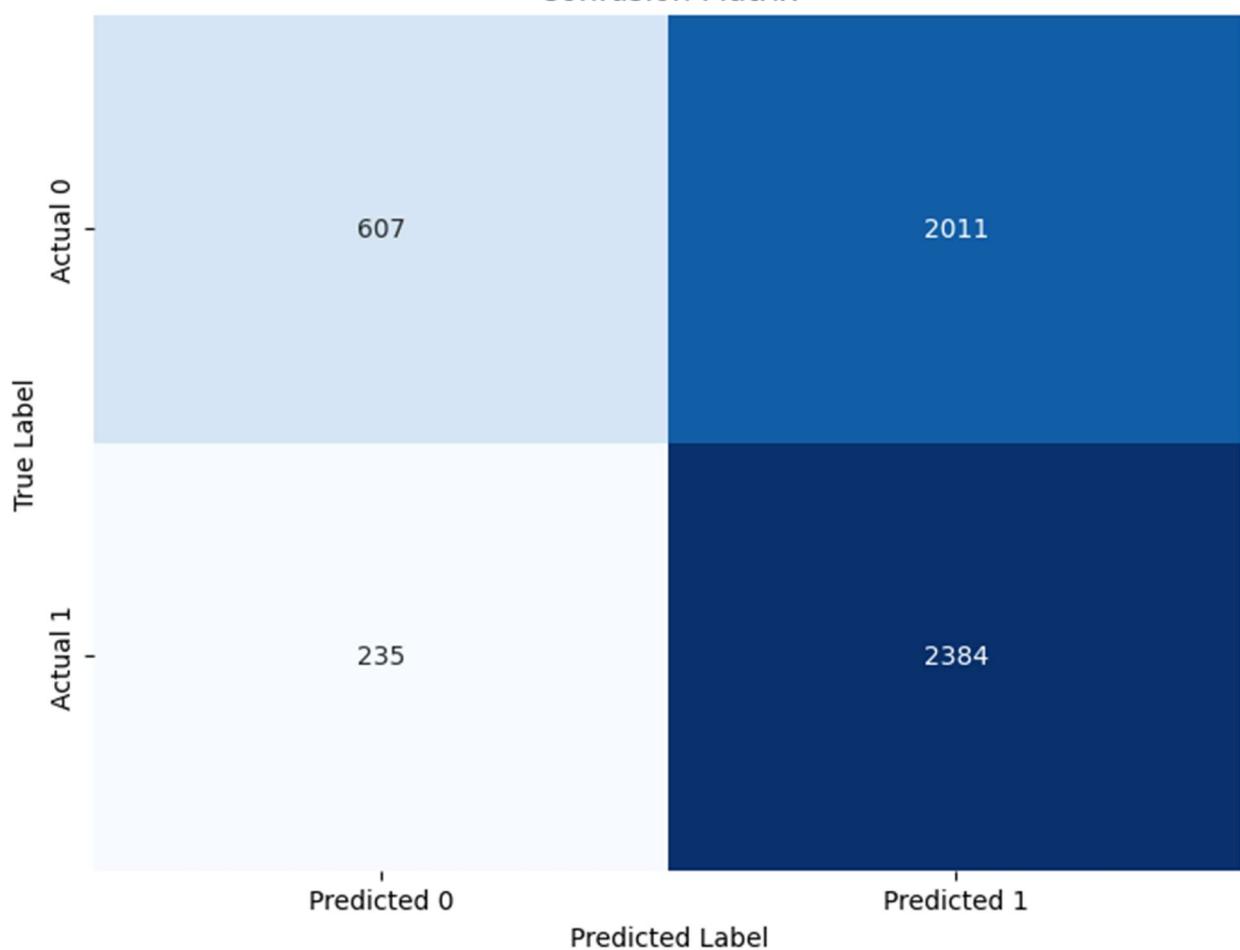
Valori Mancanti nel Dataset 10%-50%,70%

Di seguito i risultati a 10% di sporcatura:

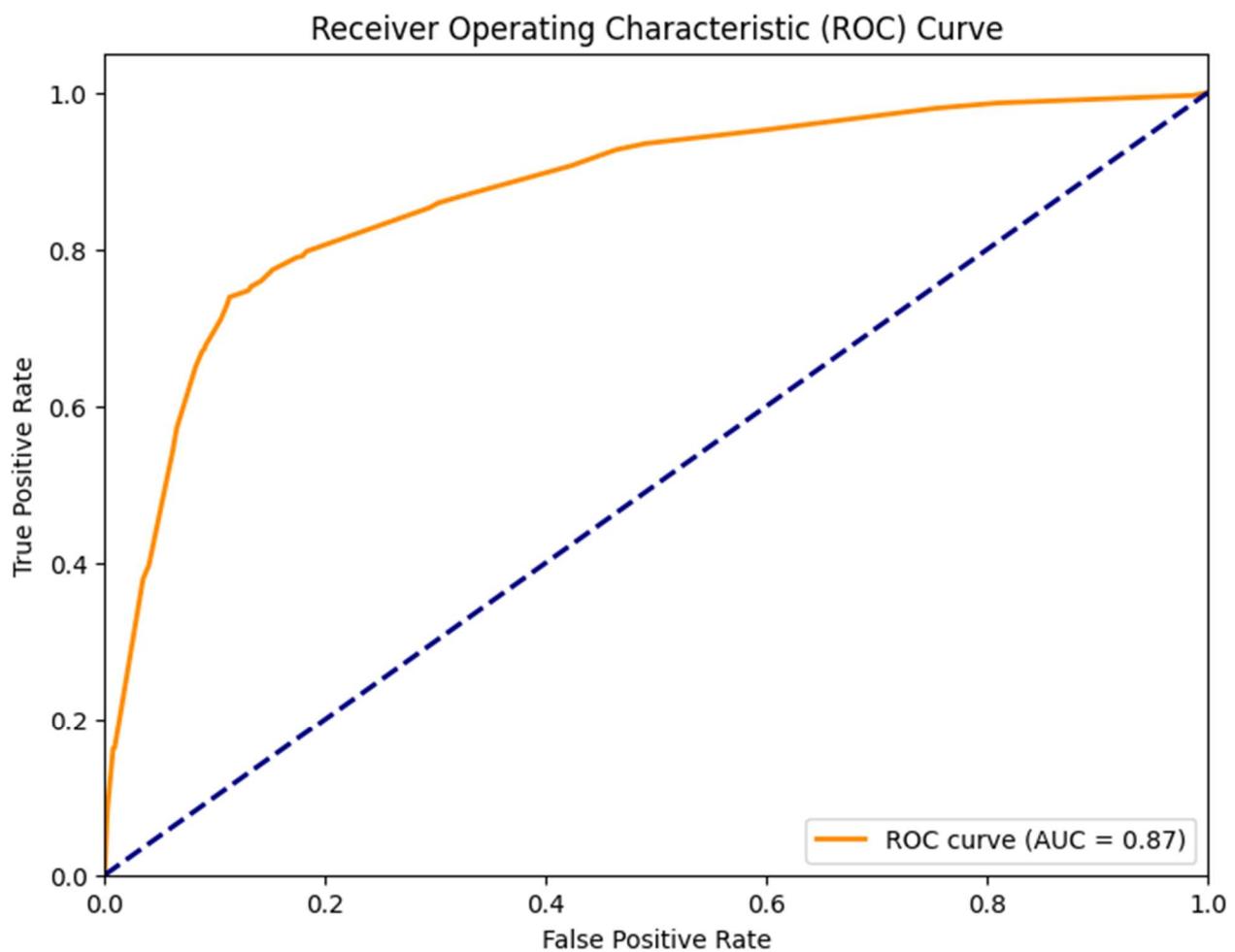
```
MIA added:  
numerical: 8573  
strings: 4867  
  
Model: GaussianNB  
Accuracy test: 0.5711 | Accuracy Training: 0.5685  
F1-score test: 0.6798 | F1-score Training: 0.6805  
roc_AUC-score test: 0.7288 | roc_AUC-score Training: 0.7127  
Differenza Accuracy: -0.0026  
Differenza F1-score: 0.0007  
Differenza AUC: -0.0162  
  
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:  
'person_income': 0.0340  
'loan_int_rate': 0.0188  
'credit_score': 0.0060  
'loan_amnt': 0.0015  
'person_home_ownership_RENT': 7.2561e-04  
'loan_intent_HOMEIMPROVEMENT': 3.8190e-05  
'person_education_High School': 3.8190e-05  
'loan_intent_EDUCATION': -2.2204e-17  
'person_home_ownership_OWN': -3.8190e-05  
'loan_intent_PERSONAL': -3.8190e-05  
'person_education_Bachelor': -3.8190e-05  
'loan_intent_VENTURE': -7.6380e-05  
'cb_person_cred_hist_length': -0.0034  
'person_age': -0.0060  
'person_emp_exp': -0.0062  
AUC: 0.7288
```



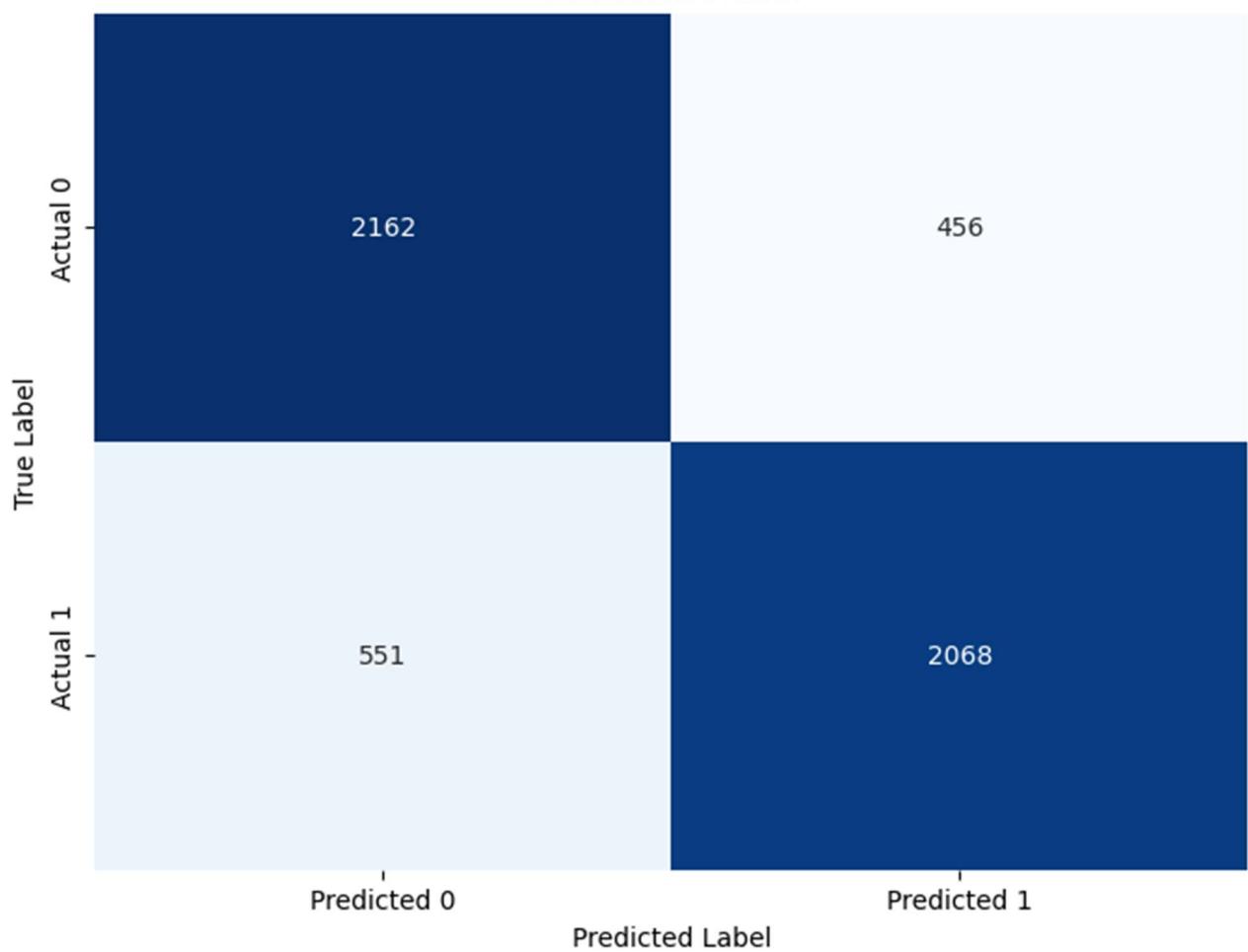
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.8077 | Accuracy Training: 0.7985
F1-score test: 0.8042 | F1-score Training: 0.7882
roc_AUC-score test: 0.8707 | roc_AUC-score Training: 0.8611
Differenza Accuracy: -0.0092
Differenza F1-score: -0.0160
Differenza AUC: -0.0096
Importanza delle Feature per Decision Tree:
'person_income': 0.3128
'loan_int_rate': 0.3083
'loan_amnt': 0.1596
'person_home_ownership_RENT': 0.0981
'person_home_ownership_OWN': 0.0375
'credit_score': 0.0354
'loan_intent_HOMEIMPROVEMENT': 0.0164
'loan_intent_VENTURE': 0.0094
'person_home_ownership_None': 0.0071
'loan_intent_EDUCATION': 0.0053
'loan_intent_PERSONAL': 0.0024
'cb_person_cred_hist_length': 0.0022
'person_emp_exp': 0.0019
'loan_intent_MEDICAL': 0.0014
'person_age': 0.0012
'person_education_Bachelor': 8.9909e-04
AUC: 0.8707
```

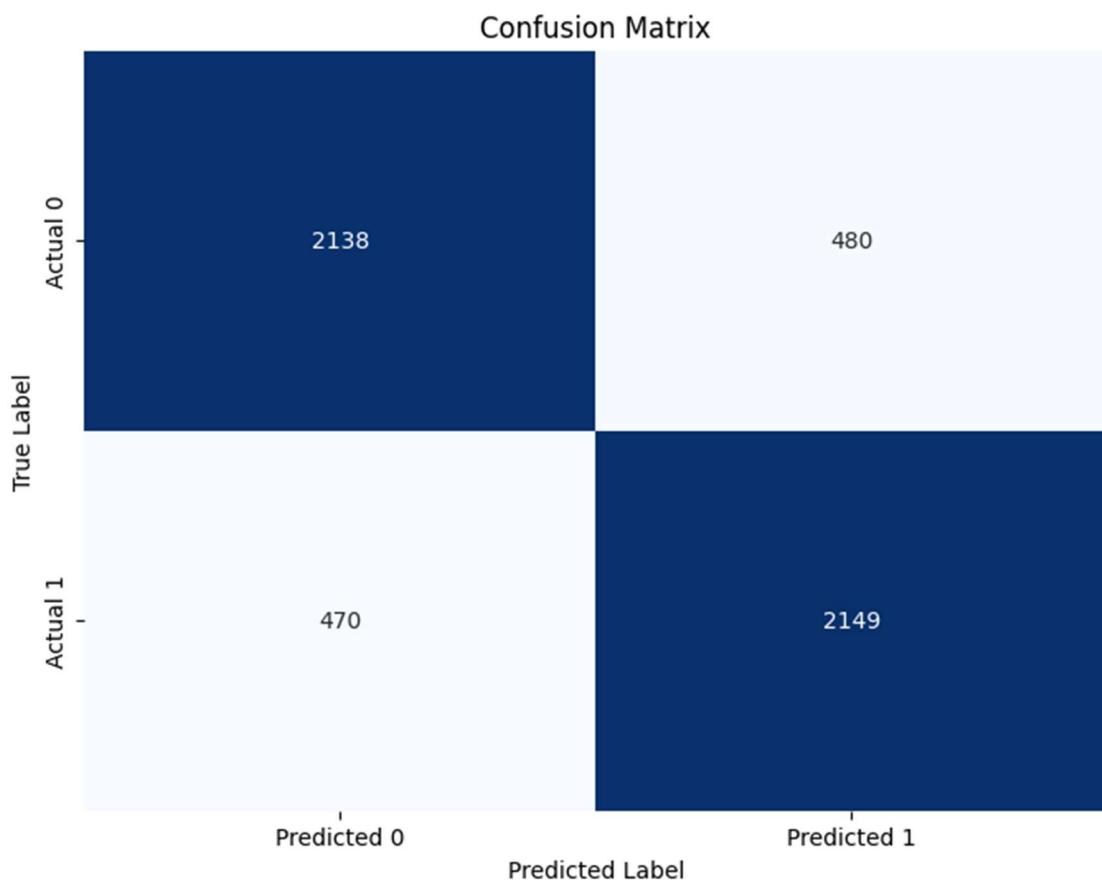
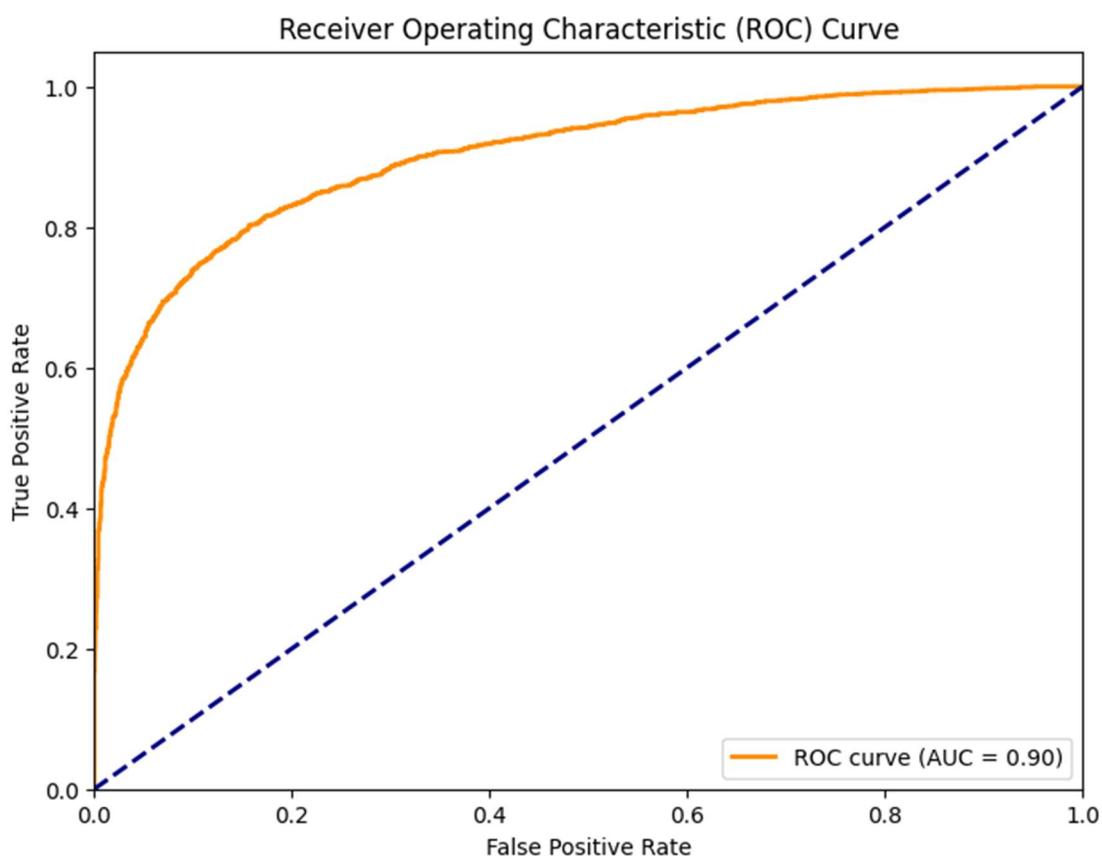


Confusion Matrix



```
Model: MLPClassifier
Accuracy test: 0.8186 | Accuracy Training: 0.8156
F1-score test: 0.8190 | F1-score Training: 0.8115
roc_AUC-score test: 0.9002 | roc_AUC-score Training: 0.8948
Differenza Accuracy: -0.0030
Differenza F1-score: -0.0075
Differenza AUC: -0.0054

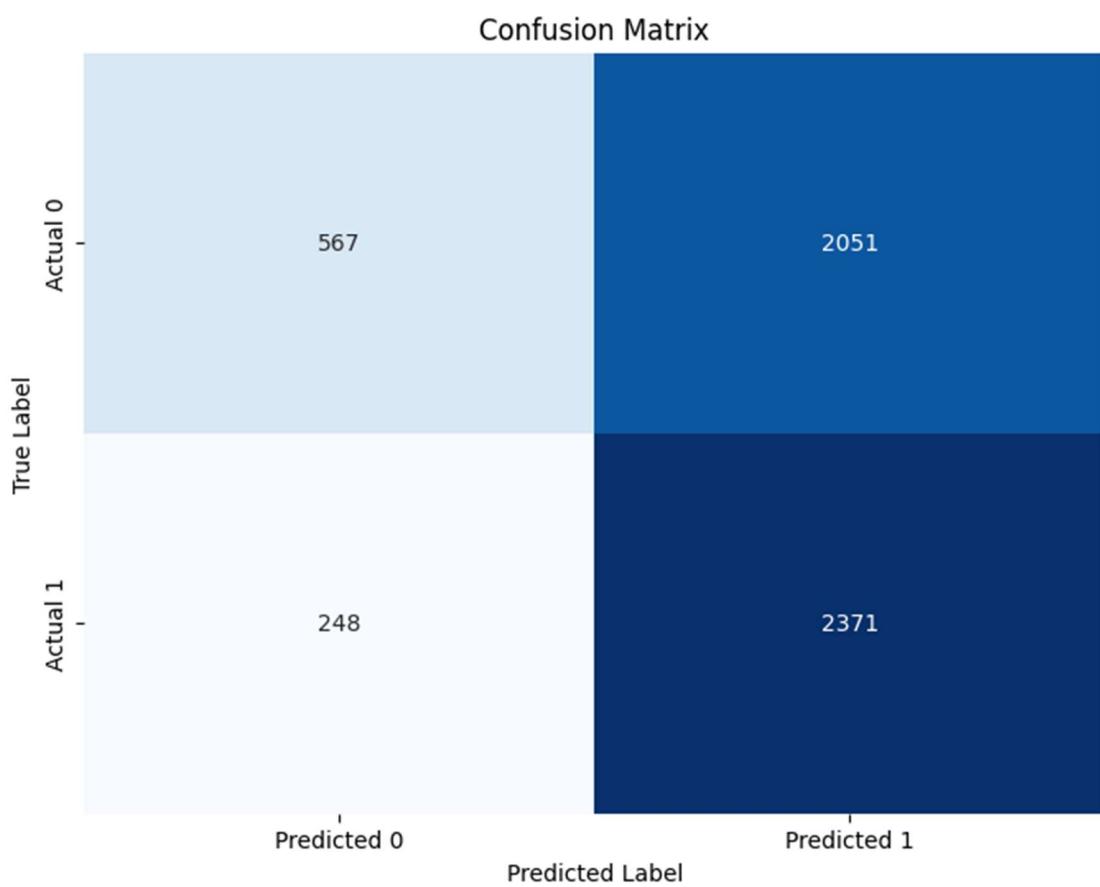
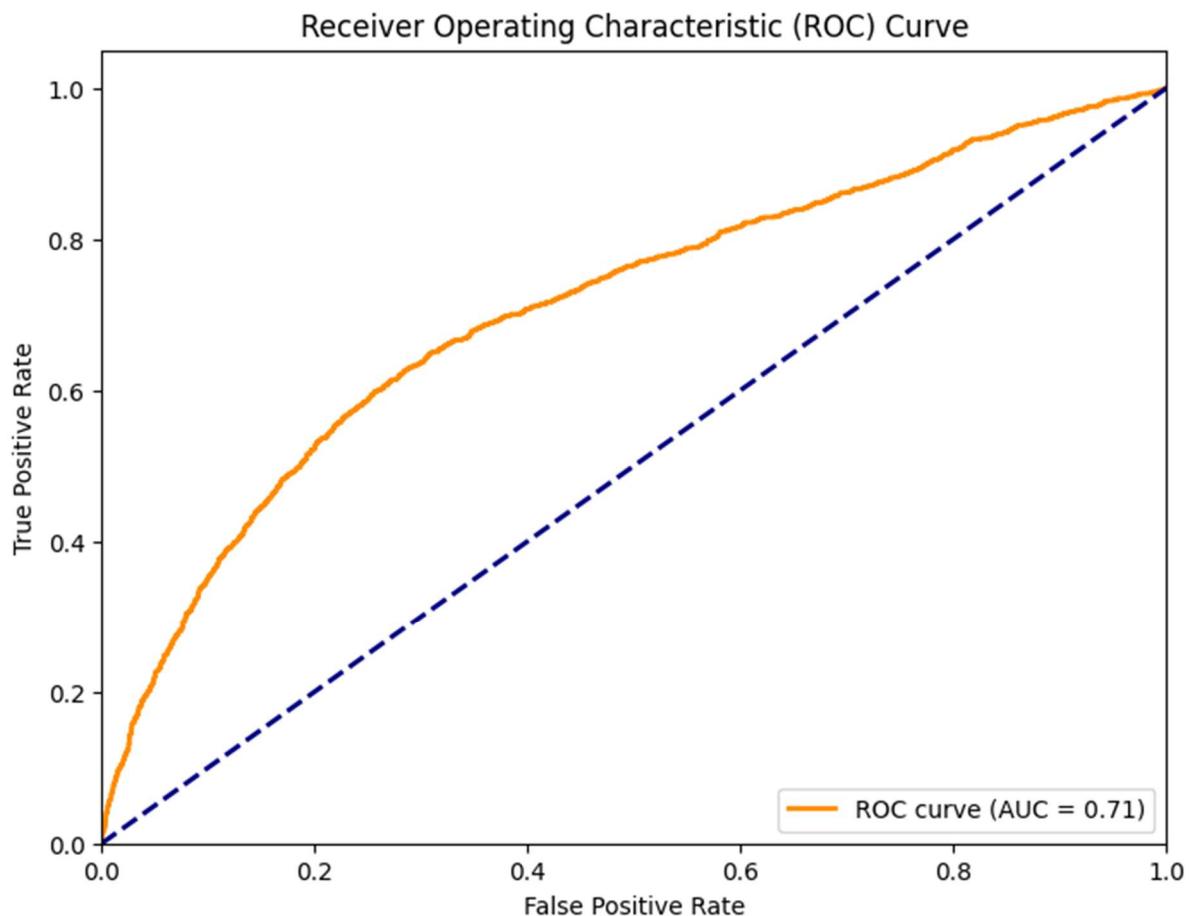
Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age 0.000000
person_emp_exp 0.000000
cb_person_cred_hist_length 0.000000
loan_int_rate 0.000000
credit_score 0.000000
person_gender_male 0.000000
person_education_Bachelor 0.000000
person_gender_None 0.000000
person_home_ownership_RENT 0.000000
person_home_ownership_None 0.000000
person_education_Doctorate 0.000000
person_education_High School 0.000000
person_education_Master 0.000000
person_education_None 0.000000
person_home_ownership_OTHER 0.000000
person_home_ownership_OWN 0.000000
loan_intent_MEDICAL 0.000000
loan_intent_PERSONAL 0.000000
loan_intent_EDUCATION 0.000000
loan_intent_HOMEIMPROVEMENT 0.000000
loan_intent_VENTURE 0.000000
loan_intent_None 0.000000
person_income -0.001031
loan_amnt -0.002597
dtype: float64
AUC: 0.9002
```



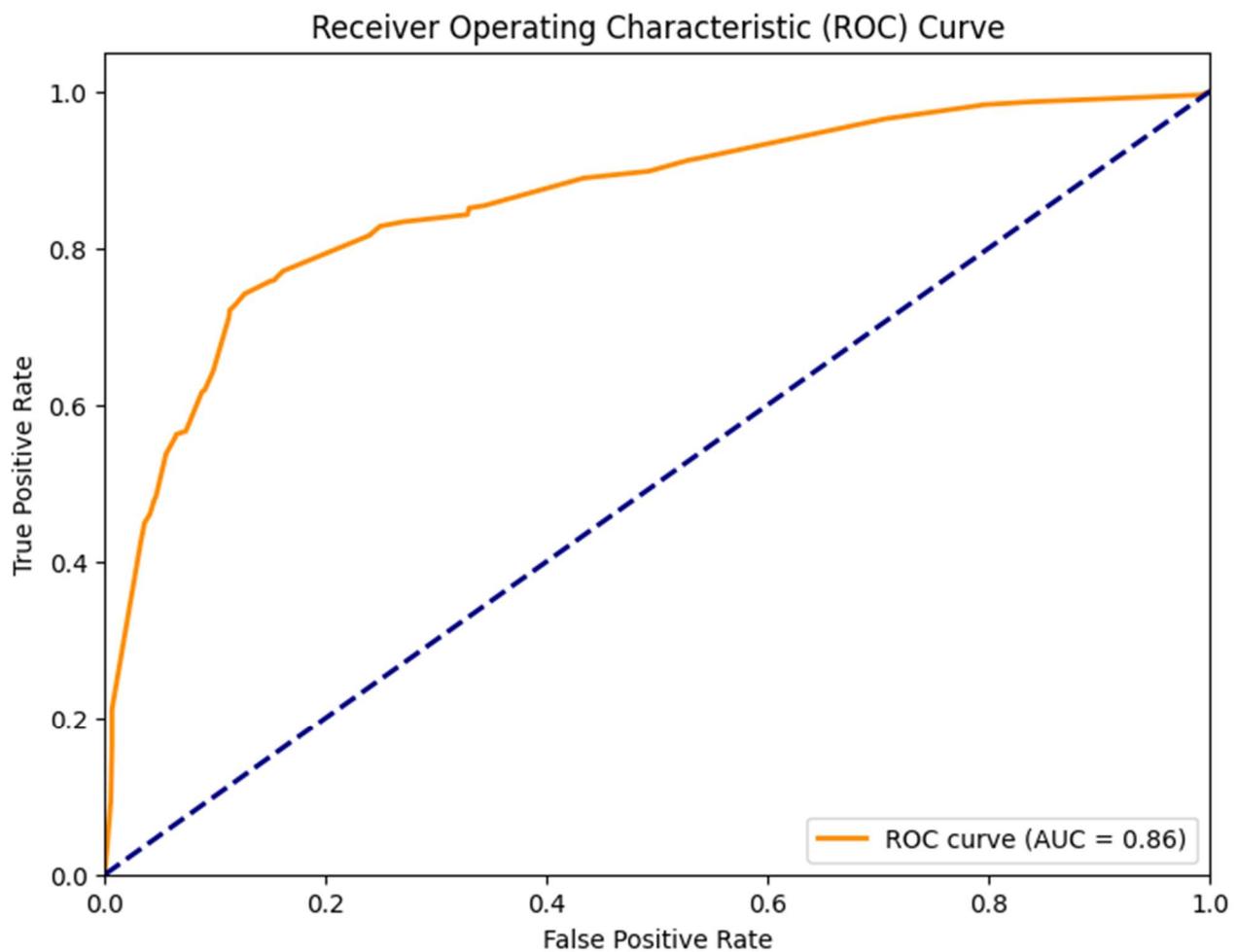
notare che, mentre nel MLP non sembrano esserci particolari cambiamenti, il Naive-bayes presenta un leggero peggioramento sia nel' AUC che nella confusion matrix ed l'albero decisionale sembra presentare dei leggeri miglioramenti sia nei valori numerici che nella distribuzione delle predizioni della confusion matrix.

Di seguito i risultati a 30% di sporcatura:

```
MIA added:  
numerical: 25793  
strings: 14529  
  
Model: GaussianNB  
Accuracy test: 0.5610 | Accuracy Training: 0.5461  
F1-score test: 0.6735 | F1-score Training: 0.6726  
roc_AUC-score test: 0.7087 | roc_AUC-score Training: 0.6685  
Differenza Accuracy: -0.0149  
Differenza F1-score: -0.0009  
Differenza AUC: -0.0402  
  
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:  
'person_income': 0.0286  
'loan_int_rate': 0.0126  
'credit_score': 0.0063  
'person_home_ownership_RENT': 0.0011  
'person_home_ownership_own': 1.9095e-04  
'loan_amnt': 1.1457e-04  
'person_education_High School': 3.8190e-05  
'person_education_Master': -7.6380e-05  
'person_education_Bachelor': -1.1457e-04  
'loan_intent_MEDICAL': -1.1457e-04  
'cb_person_cred_hist_length': -0.0048  
'person_age': -0.0066  
'person_emp_exp': -0.0071  
AUC: 0.7087
```



```
Model: DecisionTreeClassifier
Accuracy test: 0.7882 | Accuracy Training: 0.7290
F1-score test: 0.7941 | F1-score Training: 0.7172
roc_AUC-score test: 0.8599 | roc_AUC-score Training: 0.8103
Differenza Accuracy: -0.0592
Differenza F1-score: -0.0768
Differenza AUC: -0.0496
Importanza delle Feature per Decision Tree:
'loan_int_rate': 0.3474
'person_income': 0.3105
'loan_amnt': 0.1392
'person_home_ownership_RENT': 0.1020
'credit_score': 0.0340
'person_home_ownership_None': 0.0180
'loan_intent_HOMEIMPROVEMENT': 0.0132
'person_home_ownership_OWN': 0.0110
'loan_intent_VENTURE': 0.0086
'person_age': 0.0051
'loan_intent_EDUCATION': 0.0033
'loan_intent_PERSONAL': 0.0025
'cb_person_cred_hist_length': 0.0014
'person_emp_exp': 0.0011
'person_gender_male': 0.0011
'person_education_High School': 0.0010
'person_education_None': 8.1446e-04
AUC: 0.8599
```



Confusion Matrix

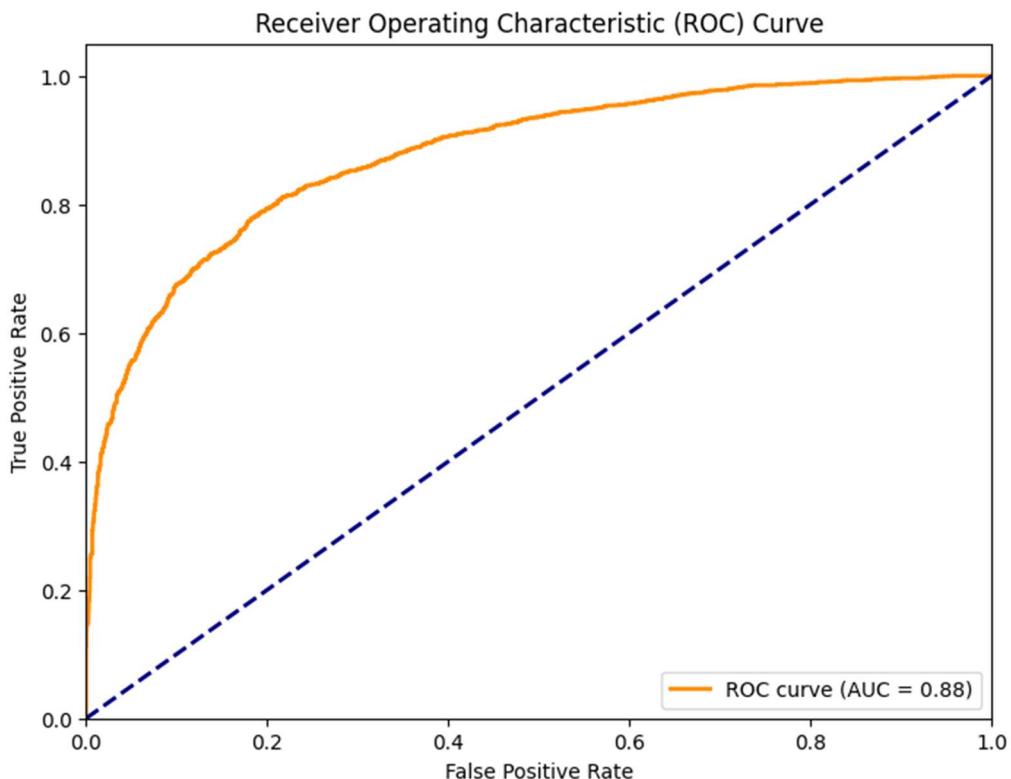
		Predicted Label
True Label	Actual 0	Actual 1
	Predicted 0	Predicted 1
Actual 0	1990	628
Actual 1	481	2138

```

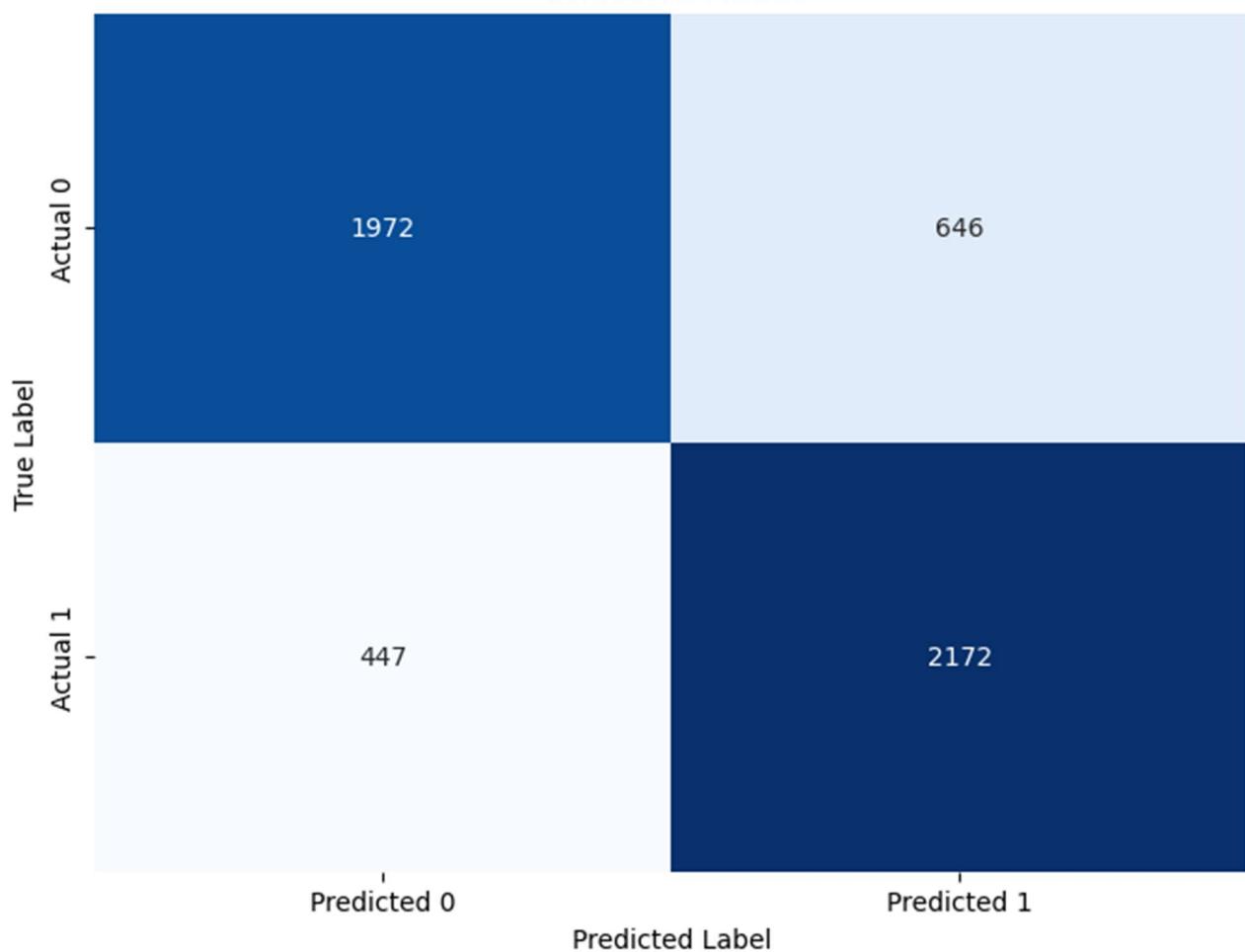
Model: MLPClassifier
Accuracy test: 0.7913 | Accuracy Training: 0.7604
F1-score test: 0.7990 | F1-score Training: 0.7579
roc_AUC-score test: 0.8773 | roc_AUC-score Training:      0.8456
Differenza Accuracy: -0.0309
Differenza F1-score: -0.0411
Differenza AUC:      -0.0317

Importanza delle Feature (stimata con Permutation Importance) per MLP
person_income          0.000038
person_age              0.000000
person_emp_exp          0.000000
loan_int_rate            0.000000
credit_score             0.000000
cb_person_cred_hist_length 0.000000
person_gender_male       0.000000
person_gender_None        0.000000
person_home_ownership_RENT 0.000000
person_education_Bachelor 0.000000
person_education_Doctorate 0.000000
person_education_High School 0.000000
person_education_Master   0.000000
person_education_None     0.000000
person_home_ownership_OTHER 0.000000
person_home_ownership_OWN 0.000000
loan_intent_MEDICAL      0.000000
person_home_ownership_None 0.000000
loan_intent_EDUCATION    0.000000
loan_intent_HOMEIMPROVEMENT 0.000000
loan_intent_VENTURE       0.000000
loan_intent_PERSONAL      0.000000
loan_intent_None           0.000000
loan_amnt                 -0.000076
dtype: float64
AUC: 0.8773

```



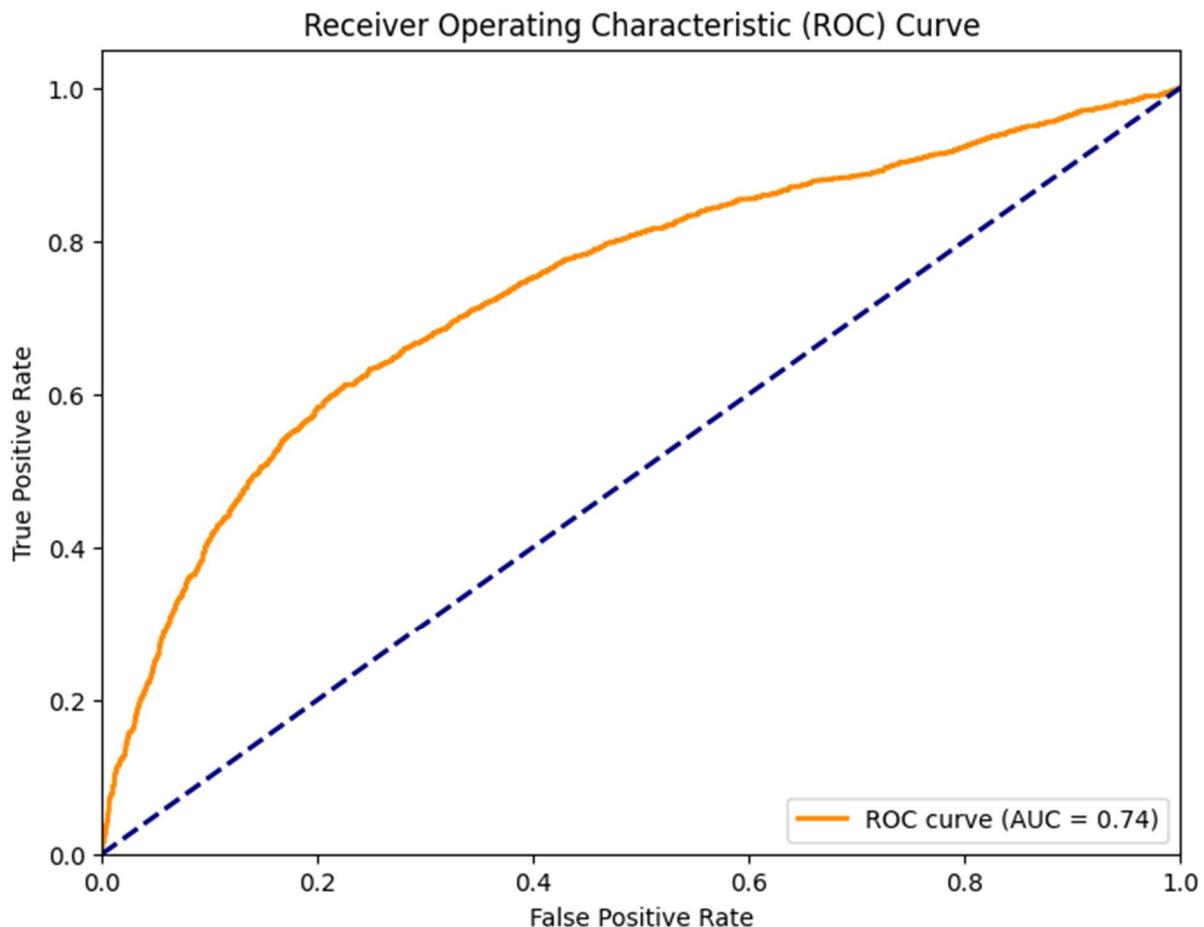
Confusion Matrix



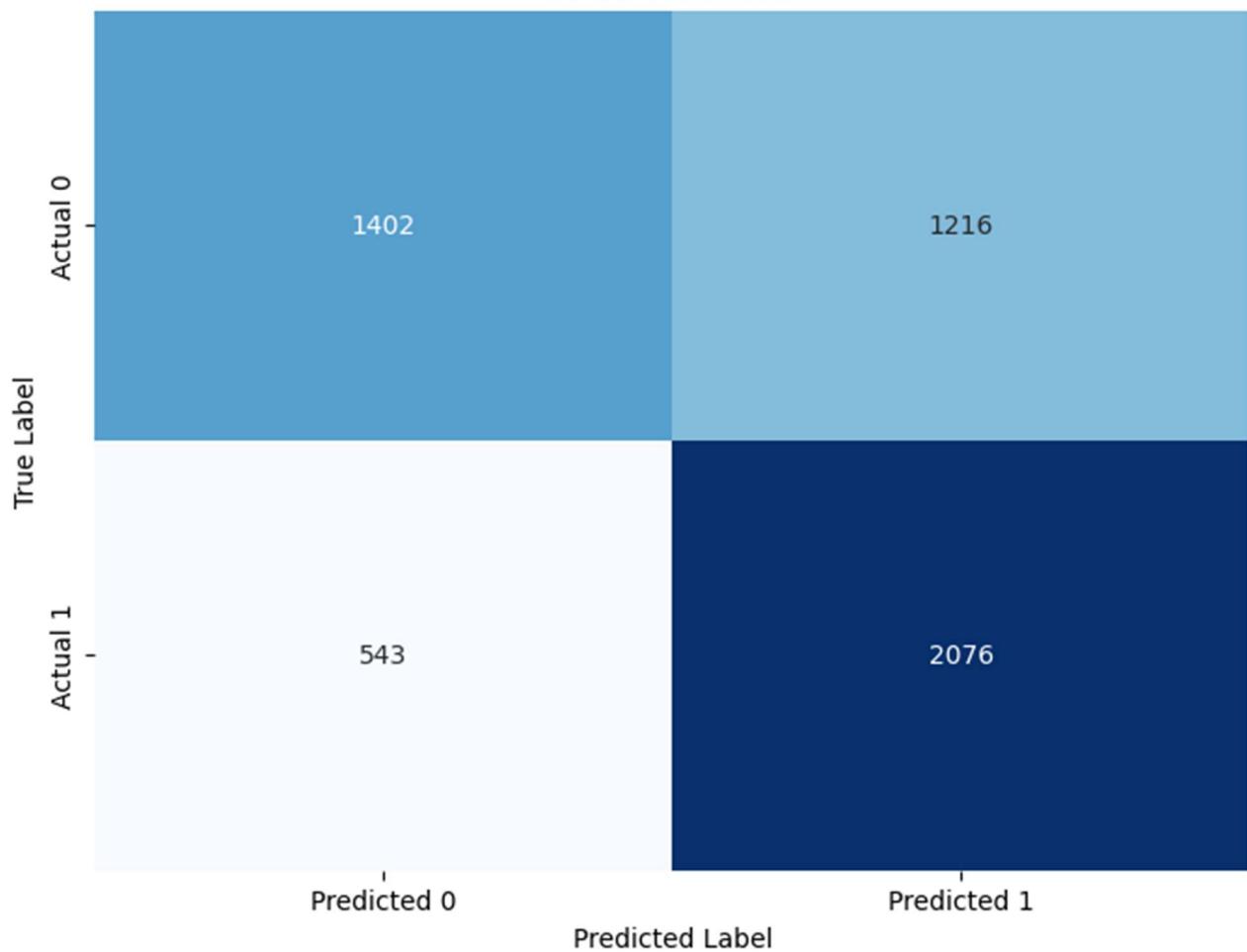
Al 30% di sporcatura notiamo come il naive-bayes continua a peggiorare drasticamente però sia il decision tree che il MLPClassifier rimangono costanti salvo delle leggere fluttuazioni dei valori probabilmente attribuibili al caso.

Di seguito i risultati a 40% di sporcatura:

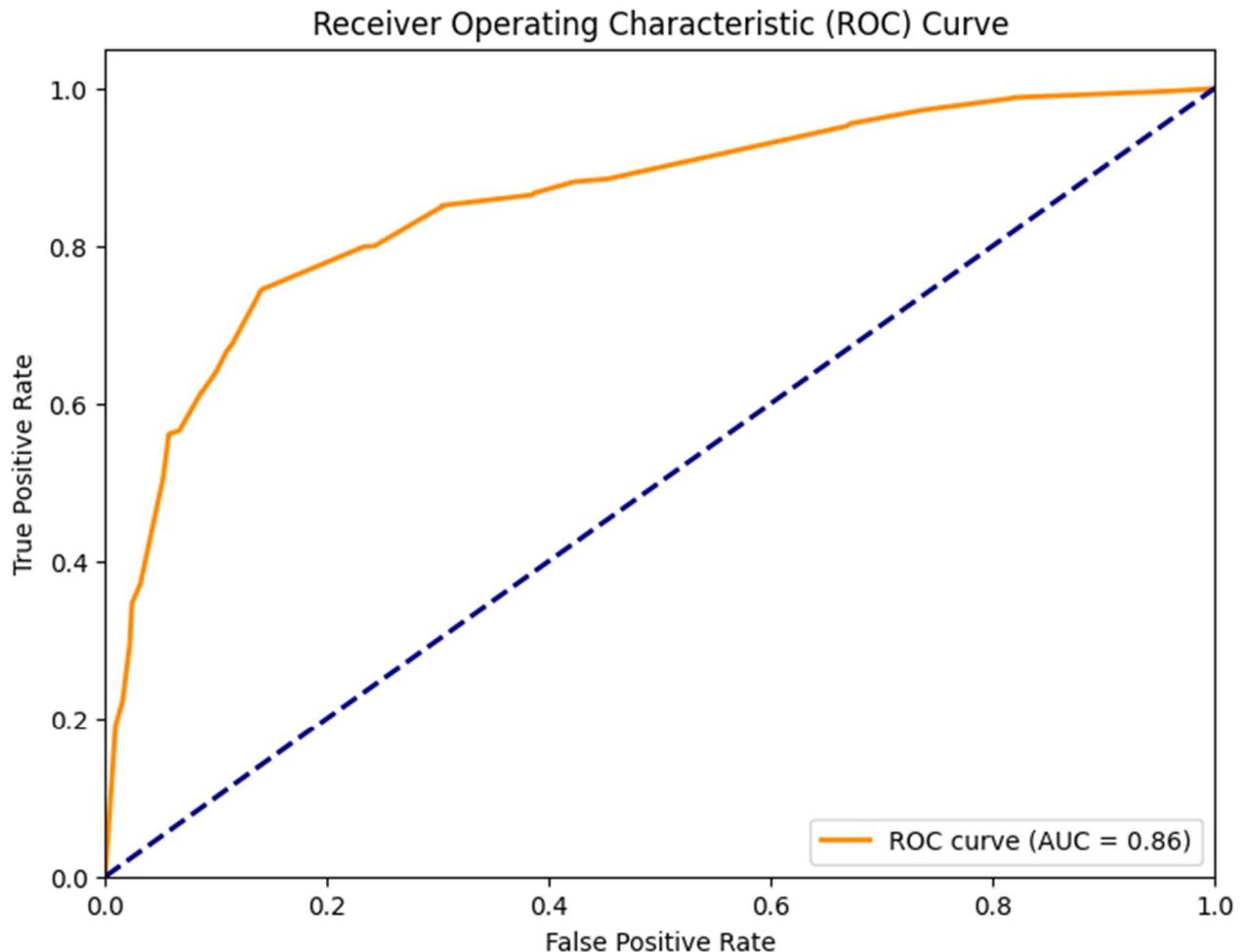
```
MIA added:  
numerical: 34199  
strings: 19564  
  
Model: GaussianNB  
Accuracy test: 0.6641 | Accuracy Training: 0.6103  
F1-score test: 0.7024 | F1-score Training: 0.6765  
roc_AUC-score test: 0.7406 | roc_AUC-score Training: 0.6838  
Differenza Accuracy: -0.0538  
Differenza F1-score: -0.0259  
Differenza AUC: -0.0568  
  
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:  
'loan_int_rate': 0.0791  
'person_income': 0.0424  
'credit_score': 0.0271  
'loan_amnt': 0.0170  
'person_home_ownership_RENT': 0.0018  
'loan_intent_HOMEIMPROVEMENT': 4.2009e-04  
'person_home_ownership_OWN': 1.9095e-04  
'loan_intent_VENTURE': 1.9095e-04  
'loan_intent_EDUCATION': 1.1457e-04  
'person_education_High School': 3.8190e-05  
'person_education_Master': -3.8190e-05  
'loan_intent_MEDICAL': -1.1457e-04  
'person_gender_male': -1.1457e-04  
'loan_intent_PERSONAL': -3.4371e-04  
'person_emp_exp': -0.0033  
'cb_person_cred_hist_length': -0.0035  
'person_age': -0.0049  
AUC: 0.7406
```



Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.7829 | Accuracy Training: 0.6983
F1-score test: 0.7863 | F1-score Training: 0.6748
roc_AUC-score test: 0.8553 | roc_AUC-score Training: 0.7844
Differenza Accuracy: -0.0846
Differenza F1-score: -0.1115
Differenza AUC: -0.0709
Importanza delle Feature per Decision Tree:
'loan_int_rate': 0.3564
'person_income': 0.3156
'loan_amnt': 0.1230
'person_home_ownership_RENT': 0.0916
'person_home_ownership_None': 0.0483
'credit_score': 0.0263
'person_home_ownership_OWN': 0.0197
'loan_intent_HOMEIMPROVEMENT': 0.0098
'loan_intent_EDUCATION': 0.0024
'loan_intent_VENTURE': 0.0023
'loan_intent_None': 0.0018
'person_age': 0.0012
'person_emp_exp': 9.0843e-04
'person_education_None': 8.7598e-04
AUC: 0.8553
```

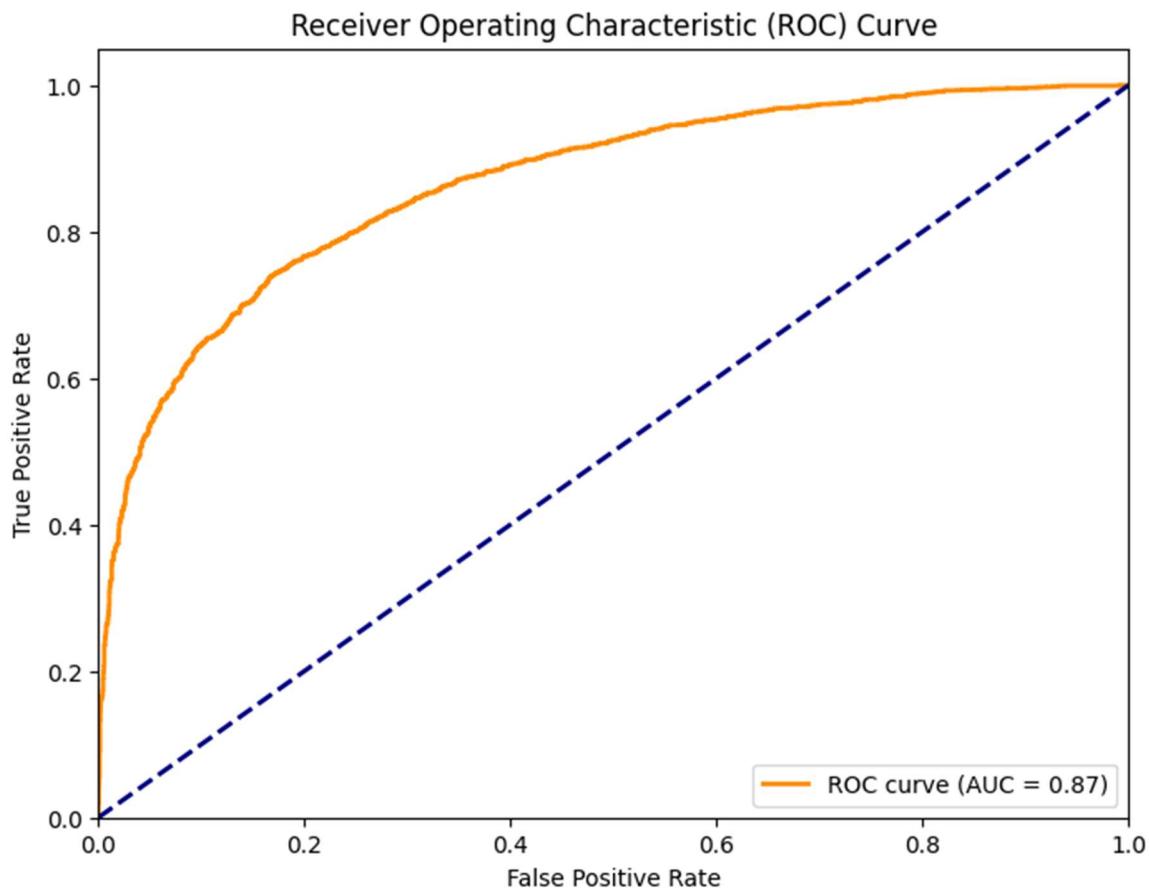


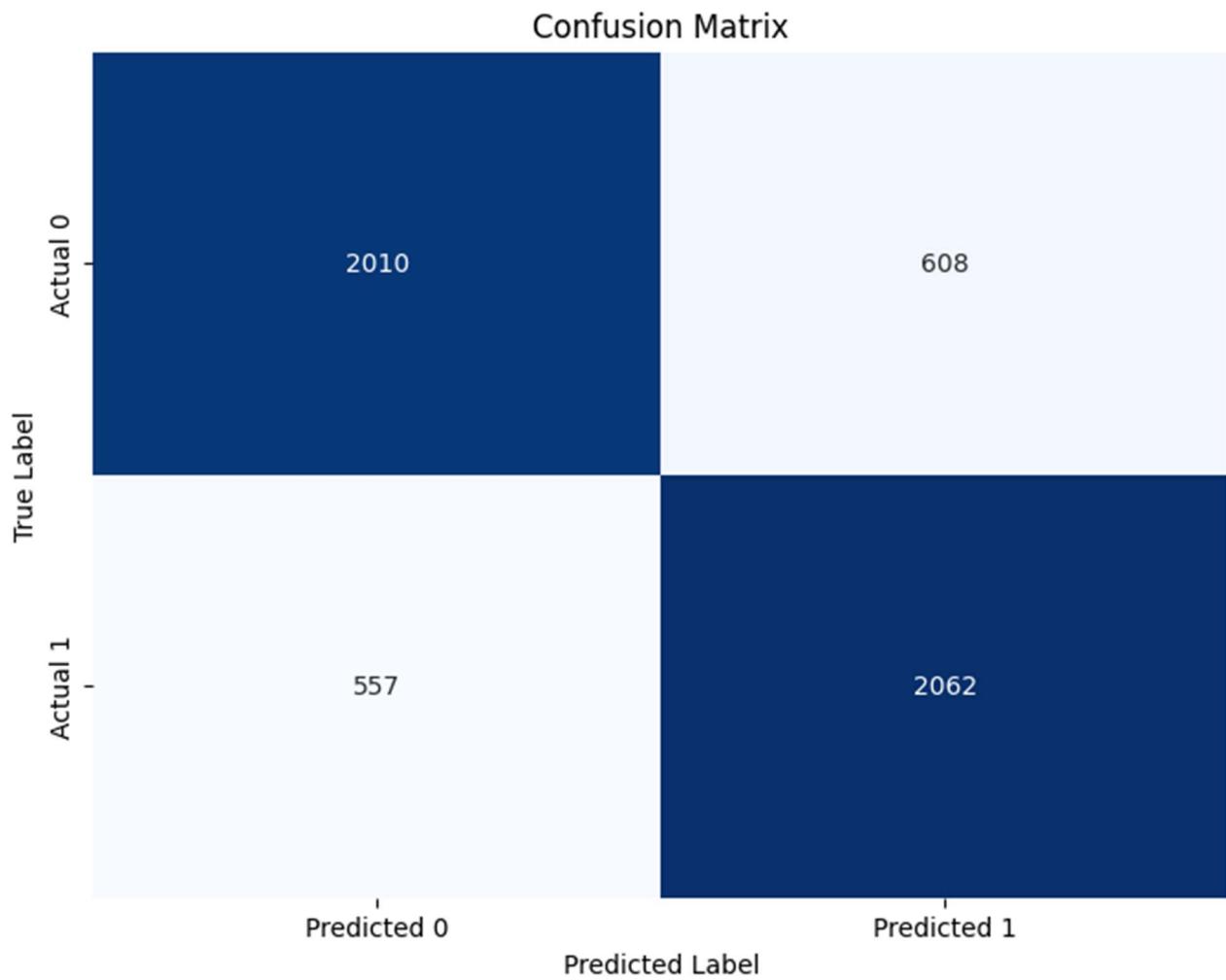
Confusion Matrix

		Predicted Label	
		Predicted 0	Predicted 1
True Label	Actual 0	2008	610
	Actual 1	527	2092

```
Model: MLPClassifier
Accuracy test: 0.7775 | Accuracy Training: 0.7370
F1-score test: 0.7797 | F1-score Training: 0.7109
roc_AUC-score test: 0.8658 | roc_AUC-score Training: 0.8285
Differenza Accuracy: -0.0405
Differenza F1-score: -0.0689
Differenza AUC: -0.0373

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age          0.000000
person_income        0.000000
person_emp_exp       0.000000
loan_int_rate        0.000000
credit_score         0.000000
cb_person_cred_hist_length 0.000000
person_gender_male   0.000000
person_gender_None   0.000000
person_home_ownership_RENT 0.000000
person_education_Bachelor 0.000000
person_education_Doctorate 0.000000
person_education_High School 0.000000
person_education_Master 0.000000
person_education_None 0.000000
person_home_ownership_OTHER 0.000000
person_home_ownership_OWN 0.000000
loan_intent_MEDICAL 0.000000
person_home_ownership_None 0.000000
loan_intent_EDUCATION 0.000000
loan_intent_HOMEIMPROVEMENT 0.000000
loan_intent_VENTURE   0.000000
loan_intent_PERSONAL 0.000000
loan_intent_None     0.000000
loan_amnt            -0.000038
dtype: float64
AUC: 0.8658
```





Con il 40% di sporcatura è bene notare come sia il decision Tree e MLP sono relativamente stabili sia numericamente che graficamente.

È interessante notare come il GaussianNB migliora drasticamente il punto di vista numerico rispetto alla sporcatura al 30% raggiungendo i valori del caso pulito ed in maniera sorprendente presenta un drastico miglioramento nella classificazione dei valori secondo la confusion matrix.

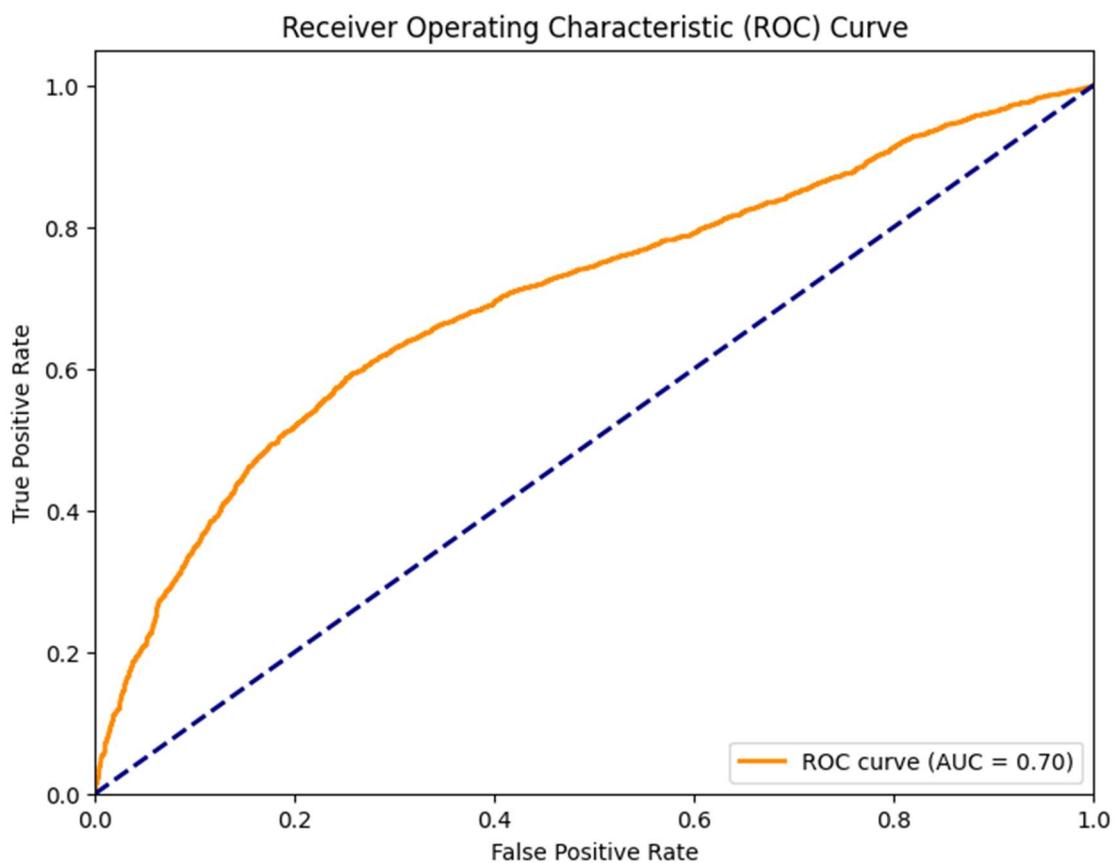
Questo miglioramento potrebbe essere causato da tre possibili fattori:

- un fattore puramente casuale
- dall'aumento dell'importanza (stimata) delle features 'loan_int_rate' (0.0791) e 'person_income' (0.0424) rispetto alle altre features
- sono stati modificati dei fattori problematici per il Naive-Bayes che causavano le basse performance originali.

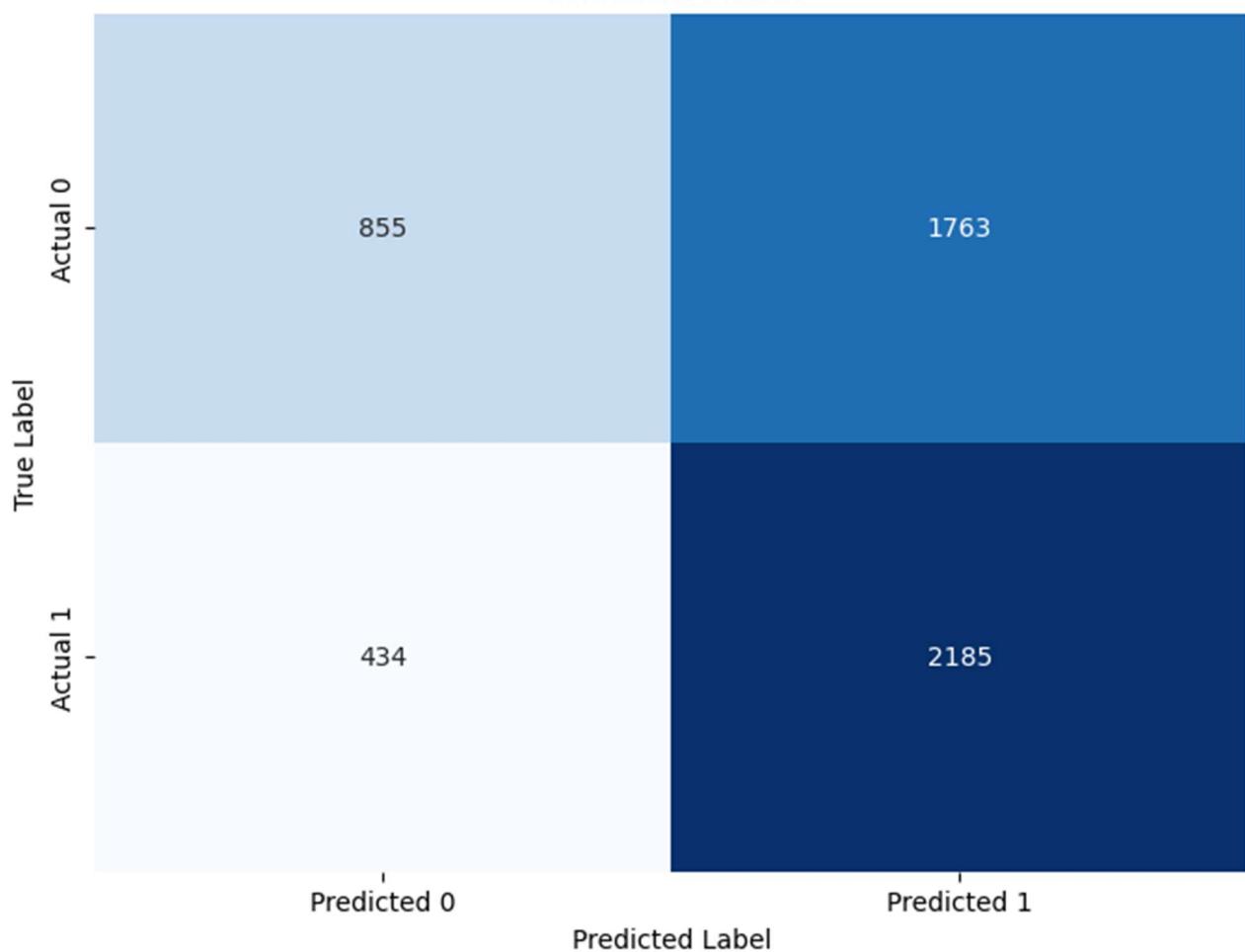
Successivamente mostrerò i valori a 50% e 70% di sporcatura, l'analisi di questi verrà fatta infine insieme

Di seguito i risultati a 50% di sporcatura:

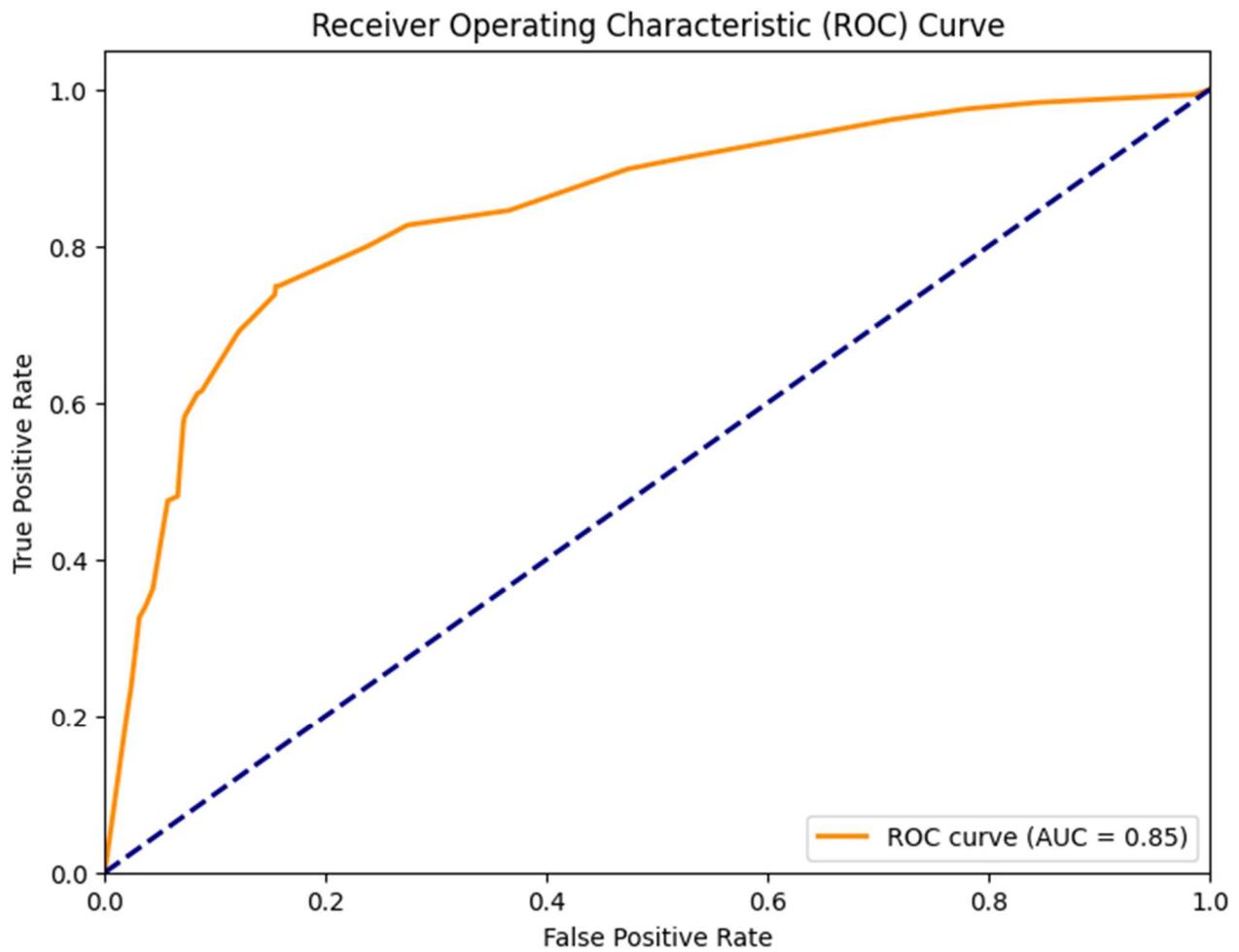
```
MIA added:  
numerical: 42772  
strings: 24432  
  
Model: GaussianNB  
Accuracy test: 0.5805 | Accuracy Training: 0.5373  
F1-score test: 0.6654 | F1-score Training: 0.6586  
roc_AUC-score test: 0.6999 | roc_AUC-score Training: 0.6325  
Differenza Accuracy: -0.0432  
Differenza F1-score: -0.0069  
Differenza AUC: -0.0674  
  
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:  
'loan_int_rate': 0.0309  
'person_income': 0.0210  
'credit_score': 0.0139  
'person_home_ownership_RENT': 0.0023  
'person_gender_male': 7.6380e-04  
'loan_intent_EDUCATION': 6.4923e-04  
'loan_amnt': 4.5828e-04  
'person_education_Bachelor': 2.6733e-04  
'person_home_ownership_OWN': 2.6733e-04  
'person_education_High School': 2.2914e-04  
'loan_intent_PERSONAL': 1.9095e-04  
'person_education_Master': 1.9095e-04  
'loan_intent_HOMEIMPROVEMENT': 1.5276e-04  
'cb_person_cred_hist_length': -2.2204e-17  
'person_education_Doctorate': -3.8190e-05  
'loan_intent_VENTURE': -1.9095e-04  
'loan_intent_MEDICAL': -3.8190e-04  
'person_age': -0.0067  
'person_emp_exp': -0.0073  
AUC: 0.6999
```



Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.7808 | Accuracy Training: 0.6879
F1-score test: 0.7850 | F1-score Training: 0.6634
roc_AUC-score test: 0.8465 | roc_AUC-score Training: 0.7504
Differenza Accuracy: -0.0928
Differenza F1-score: -0.1216
Differenza AUC: -0.0962
Importanza delle Feature per Decision Tree:
'loan_int_rate': 0.3412
'person_income': 0.3021
'loan_amnt': 0.1416
'person_home_ownership_RENT': 0.1135
'person_home_ownership_None': 0.0395
'credit_score': 0.0279
'person_home_ownership_OWN': 0.0169
'loan_intent_HOMEIMPROVEMENT': 0.0064
'loan_intent_MEDICAL': 0.0049
'person_emp_exp': 0.0033
'cb_person_cred_hist_length': 0.0013
'loan_intent_VENTURE': 0.0013
AUC: 0.8465
```



Confusion Matrix

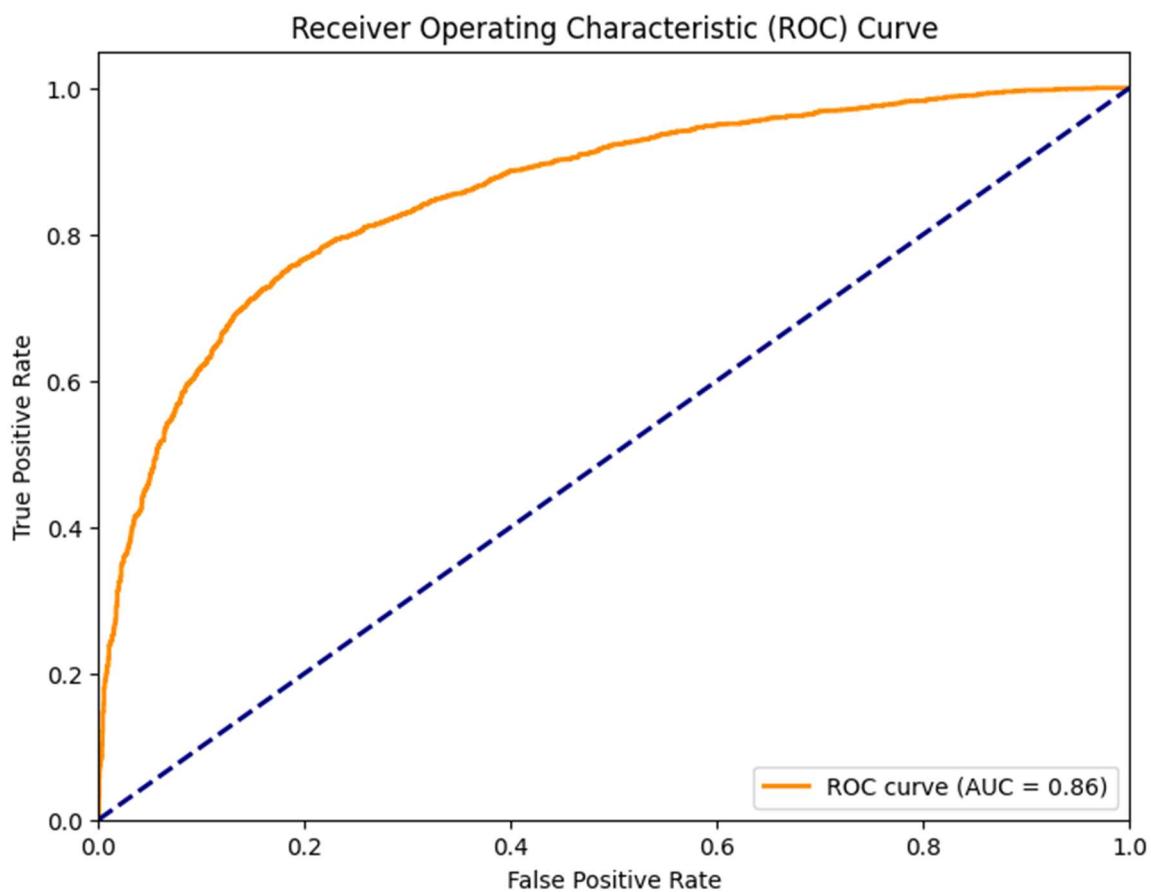
		Predicted Label
True Label	Actual 0	Actual 1
	Predicted 0	Predicted 1
Actual 0	1993	625
Actual 1	523	2096

```

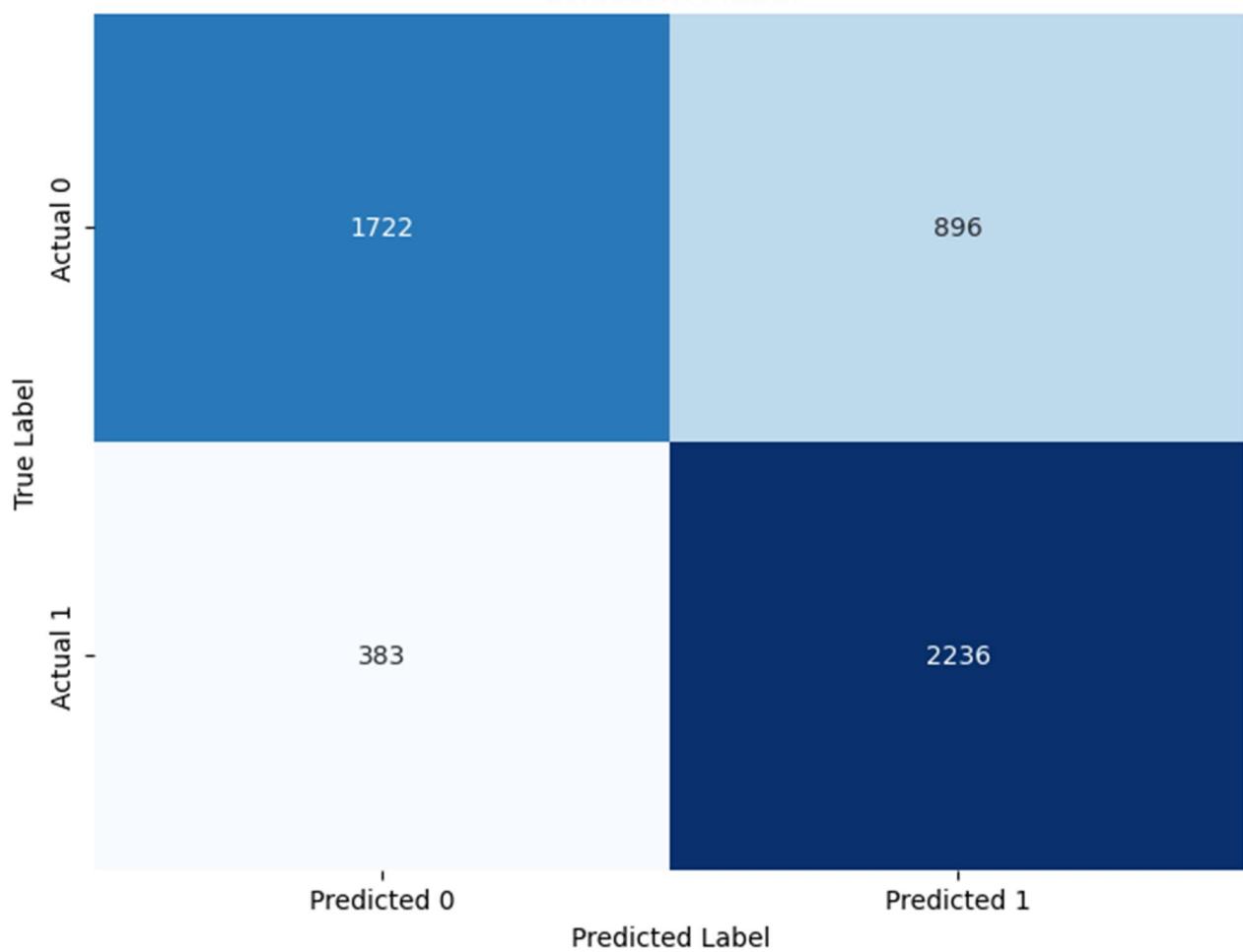
Model: MLPClassifier
Accuracy test: 0.7558 | Accuracy Training: 0.7099
F1-score test: 0.7776 | F1-score Training: 0.7101
roc_AUC-score test: 0.8564 | roc_AUC-score Training: 0.7914
Differenza Accuracy: -0.0459
Differenza F1-score: -0.0675
Differenza AUC: -0.0649

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age          0.000000
person_income        0.000000
person_emp_exp       0.000000
loan_int_rate        0.000000
credit_score         0.000000
cb_person_cred_hist_length 0.000000
person_gender_male   0.000000
person_gender_None   0.000000
person_home_ownership_RENT 0.000000
person_education_Bachelor 0.000000
person_education_Doctorate 0.000000
person_education_High School 0.000000
person_education_Master 0.000000
person_education_None 0.000000
person_home_ownership_OTHER 0.000000
person_home_ownership_OWN 0.000000
loan_intent_MEDICAL 0.000000
person_home_ownership_None 0.000000
loan_intent_EDUCATION 0.000000
loan_intent_HOMEIMPROVEMENT 0.000000
loan_intent_VENTURE    0.000000
loan_intent_PERSONAL 0.000000
loan_intent_None      0.000000
loan_amnt            -0.000038
dtype: float64
AUC: 0.8564

```



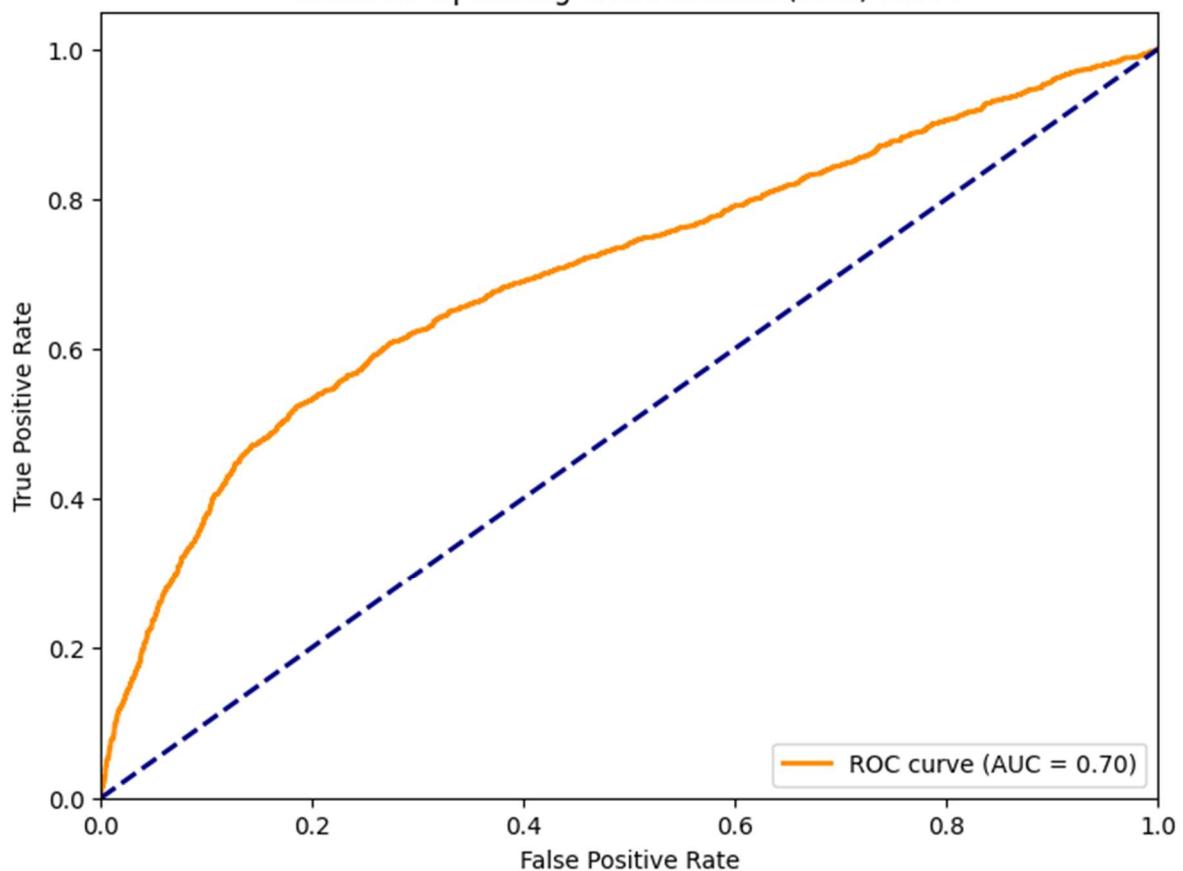
Confusion Matrix



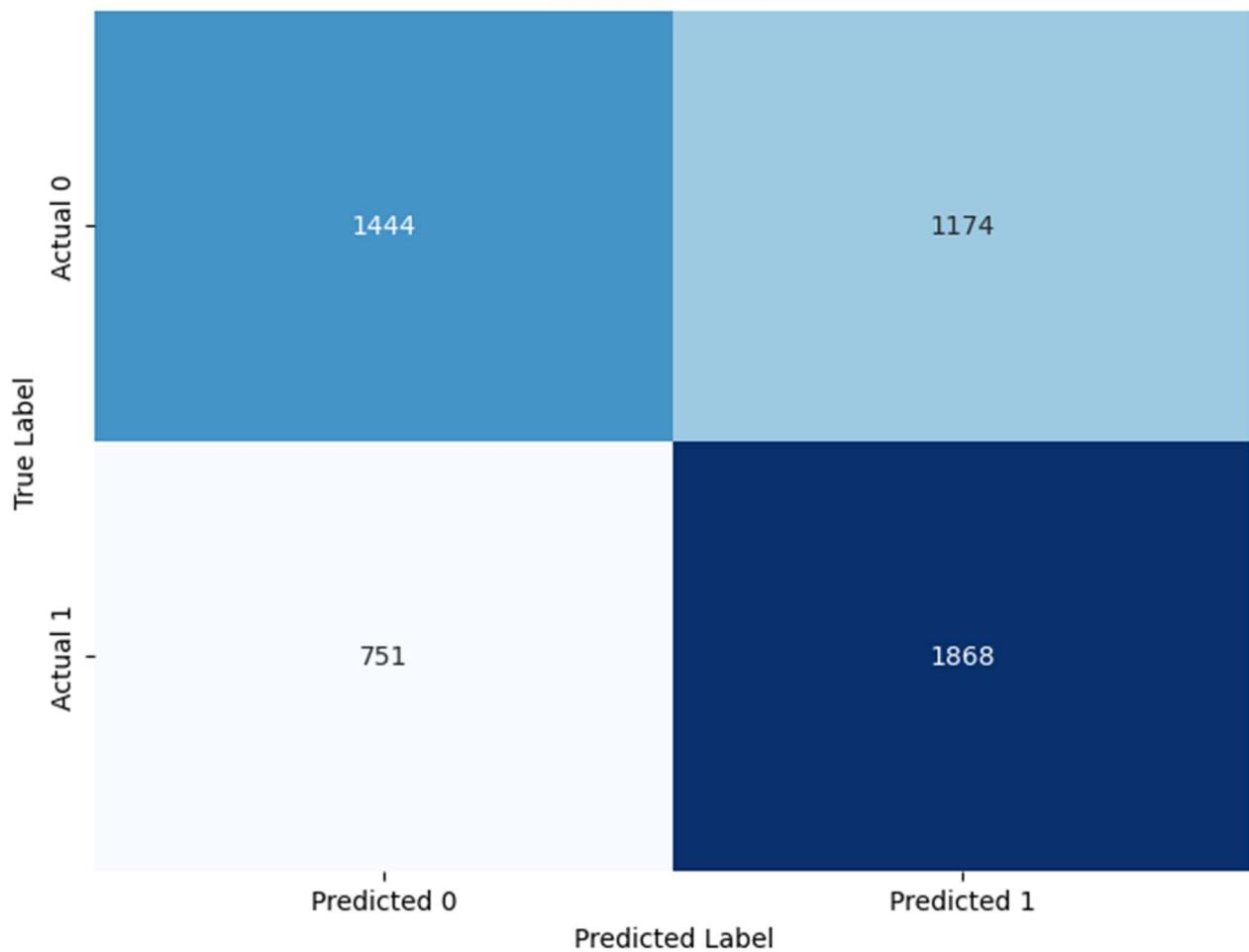
Di seguito i risultati a 70% di sporcatura:

```
MIA added:  
numerical: 59932  
strings: 34154  
  
Model: GaussianNB  
Accuracy test: 0.6324 | Accuracy Training: 0.5366  
F1-score test: 0.6600 | F1-score Training: 0.6259  
roc_AUC-score test: 0.7010 | roc_AUC-score Training: 0.5912  
Differenza Accuracy: -0.0958  
Differenza F1-score: -0.0341  
Differenza AUC: -0.1098  
  
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:  
'loan_int_rate': 0.0586  
'person_income': 0.0312  
'credit_score': 0.0225  
'loan_amnt': 0.0174  
'person_home_ownership_RENT': 0.0044  
'loan_intent_VENTURE': 5.7285e-04  
'person_education_High School': 3.4371e-04  
'person_home_ownership_OWN': 3.0552e-04  
'loan_intent_EDUCATION': 2.2914e-04  
'person_gender_male': 1.9095e-04  
'loan_intent_HOMEIMPROVEMENT': 7.6380e-05  
'person_education_Bachelor': -7.6380e-05  
'loan_intent_MEDICAL': -1.1457e-04  
'loan_intent_PERSONAL': -2.6733e-04  
'person_education_Master': -3.4371e-04  
'person_emp_exp': -5.7285e-04  
'cb_person_cred_hist_length': -0.0022  
'person_age': -0.0027  
AUC: 0.7010
```

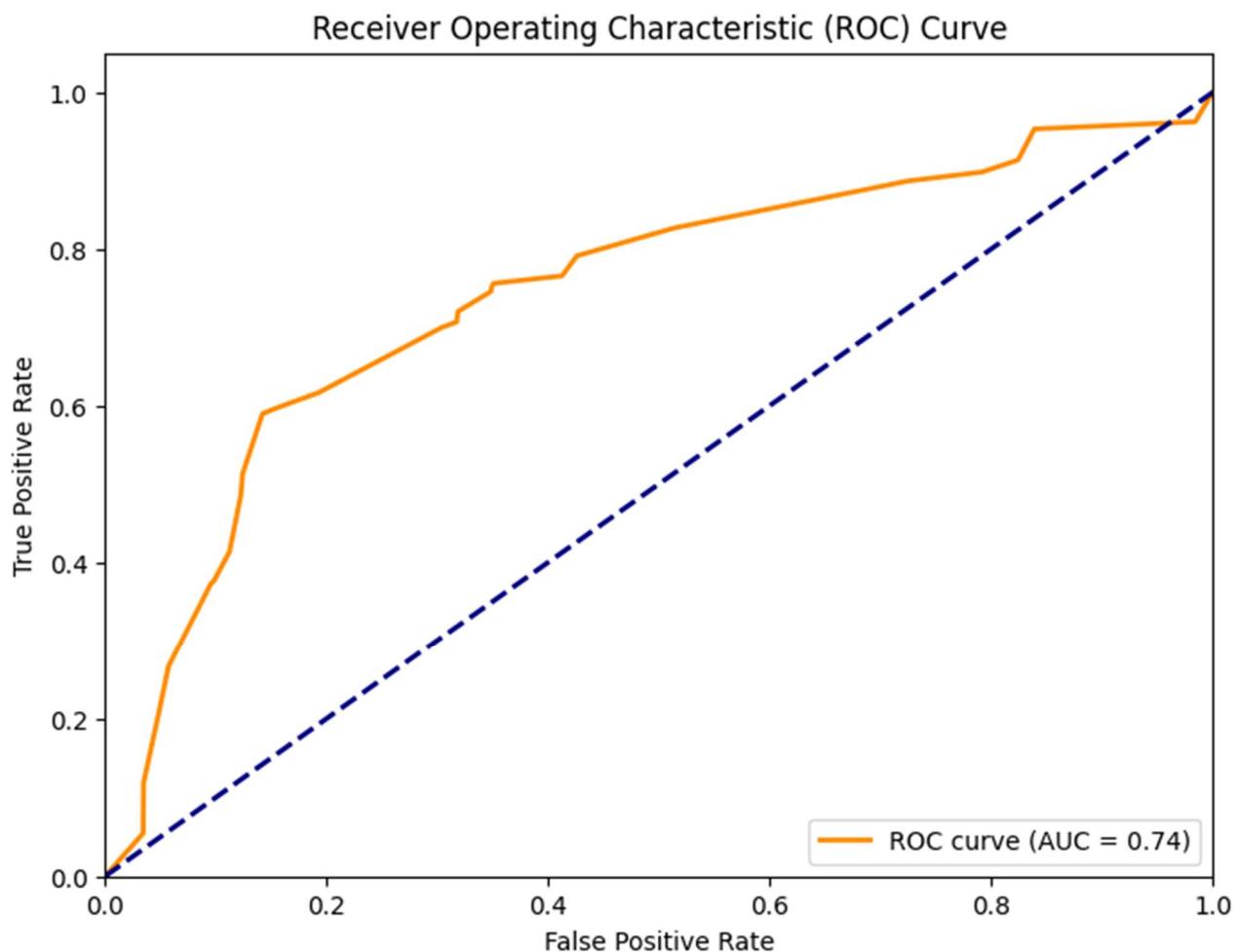
Receiver Operating Characteristic (ROC) Curve



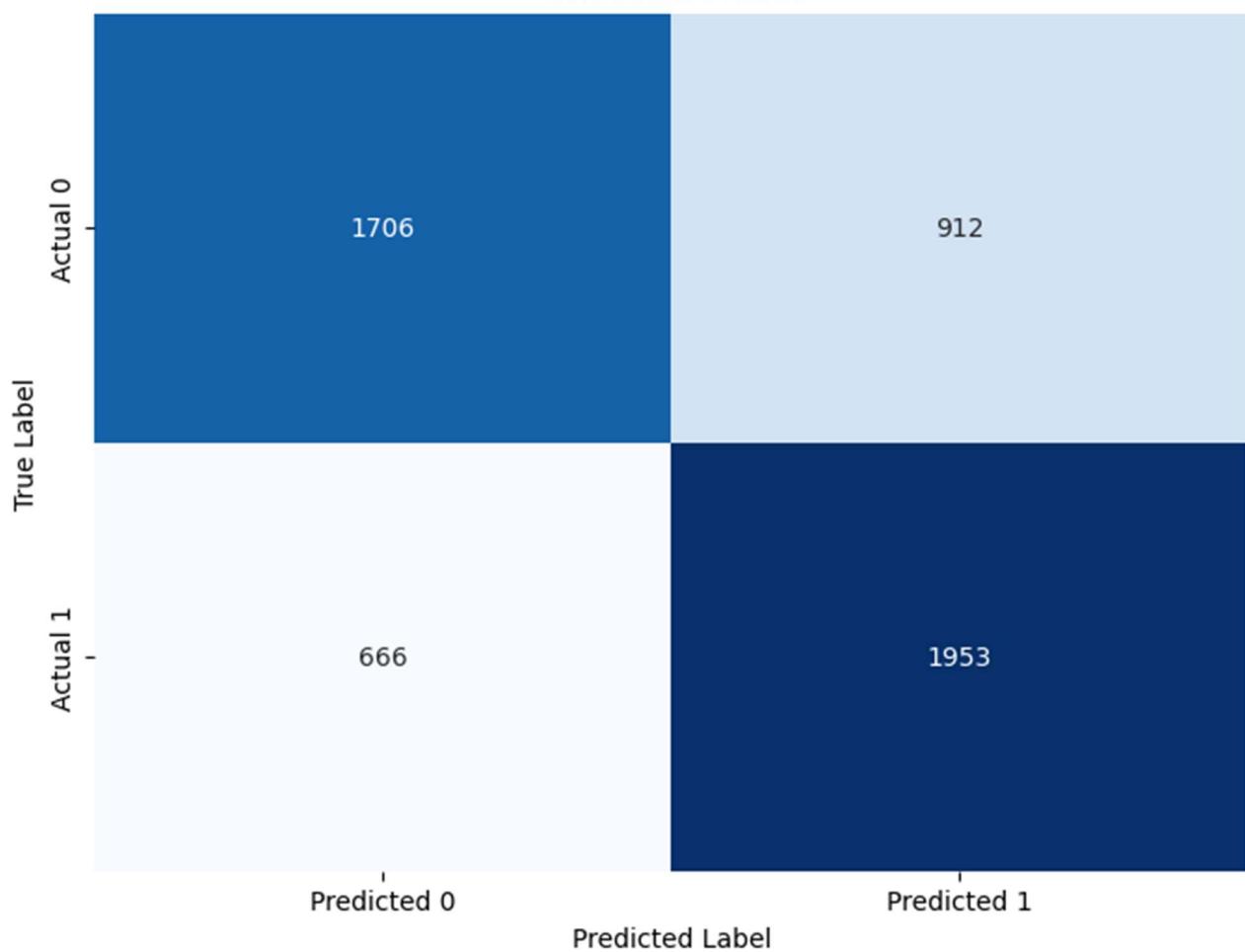
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.6987 | Accuracy Training: 0.6095
F1-score test: 0.7123 | F1-score Training: 0.4870
roc_AUC-score test: 0.7438 | roc_AUC-score Training: 0.6724
Differenza Accuracy: -0.0891
Differenza F1-score: -0.2252
Differenza AUC: -0.0714
Importanza delle Feature per Decision Tree:
'loan_int_rate': 0.3760
'person_income': 0.3026
'person_home_ownership_None': 0.0899
'person_home_ownership_RENT': 0.0872
'loan_amnt': 0.0613
'credit_score': 0.0251
'loan_intent_MEDICAL': 0.0211
'person_age': 0.0095
'person_home_ownership_OWN': 0.0081
'cb_person_cred_hist_length': 0.0063
'loan_intent_None': 0.0042
'person_emp_exp': 0.0038
'loan_intent_VENTURE': 0.0035
'loan_intent_PERSONAL': 0.0014
AUC: 0.7438
```



Confusion Matrix

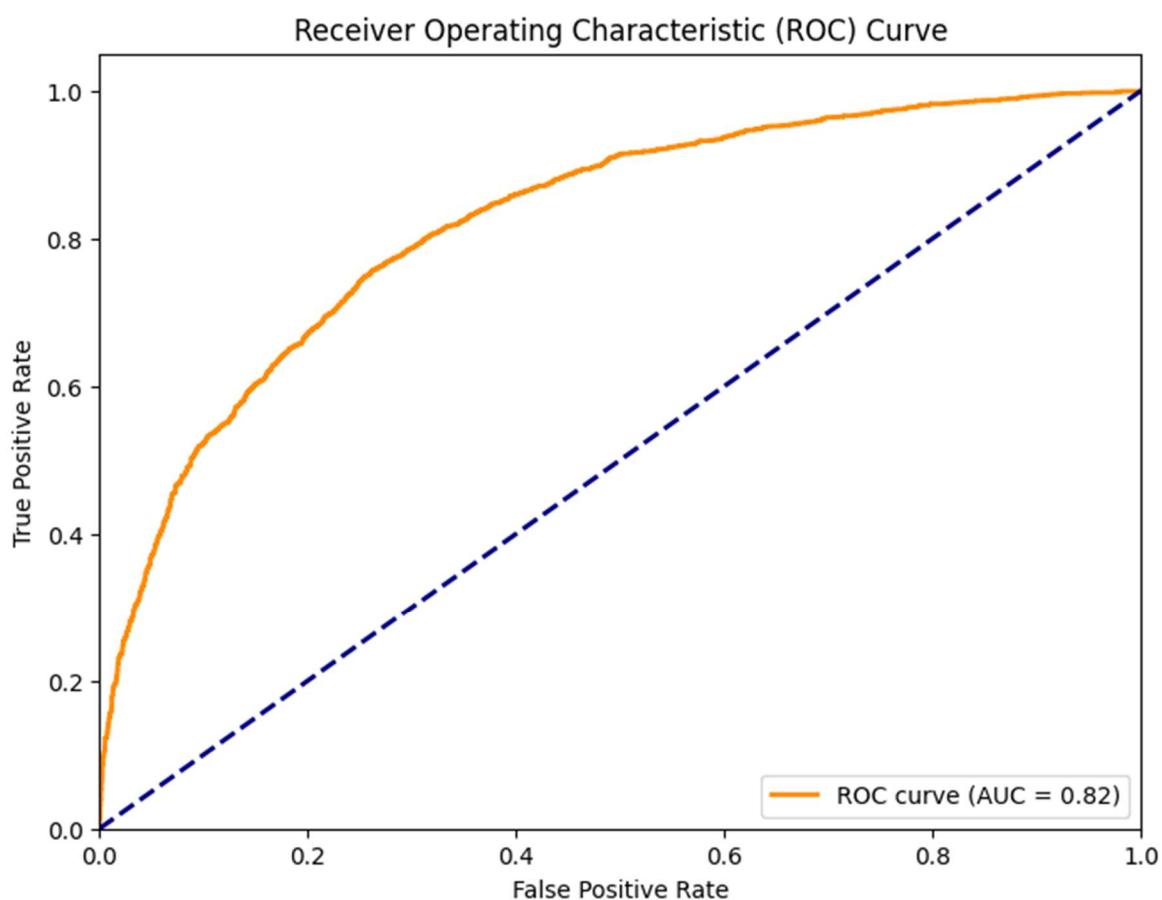


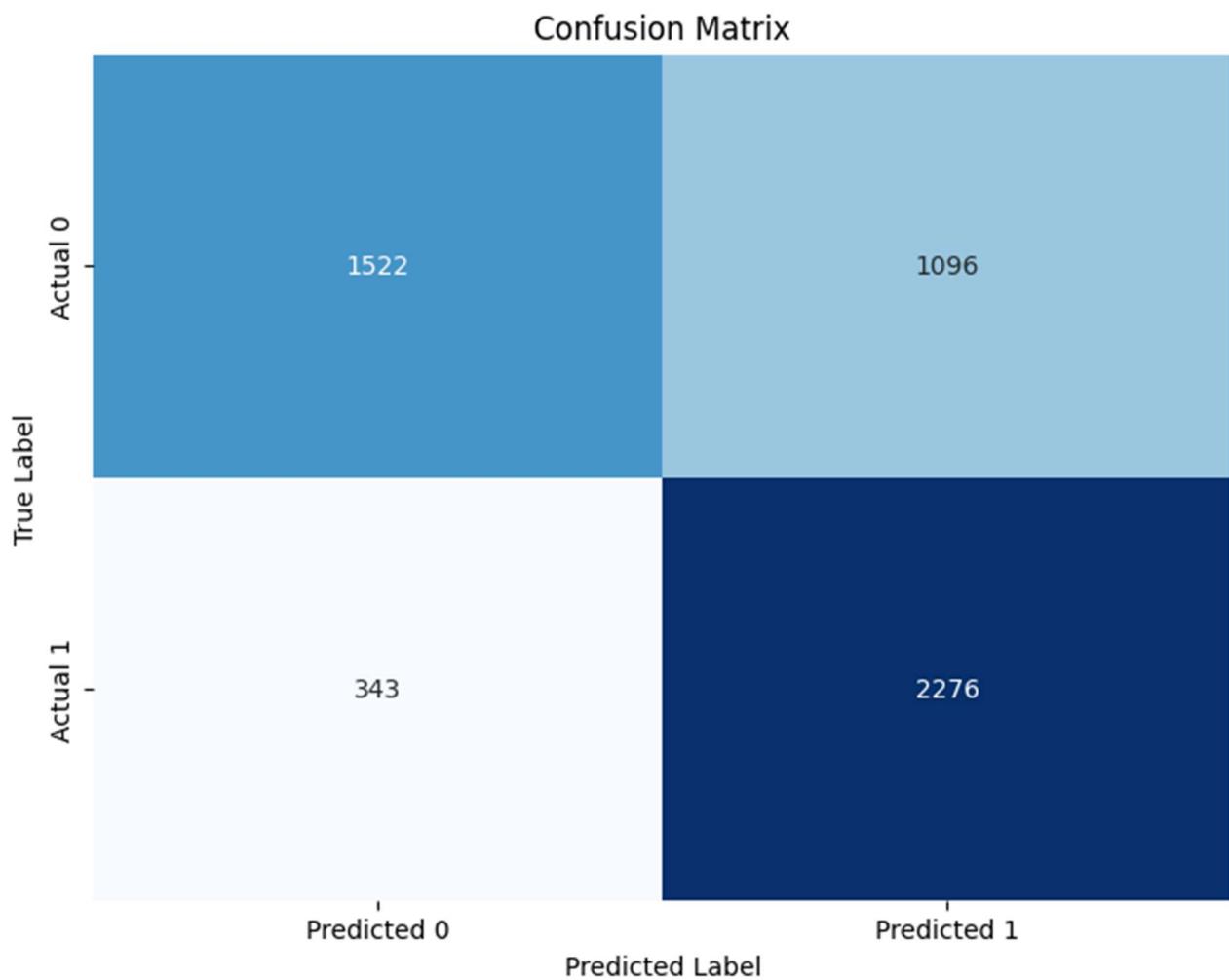
```

Model: MLPClassifier
Accuracy test: 0.7252 | Accuracy Training: 0.6599
F1-score test: 0.7598 | F1-score Training: 0.6754
roc_AUC-score test: 0.8222 | roc_AUC-score Training: 0.7358
Differenza Accuracy: -0.0654
Differenza F1-score: -0.0844
Differenza AUC: -0.0864

Importanza delle Feature (stimata con Permutation Importance) per MLP:
loan_amnt           0.003628
cb_person_cred_hist_length 0.000153
person_home_ownership_RENT 0.000115
person_gender_male     0.000076
person_education_High School 0.000038
loan_intent_MEDICAL    0.000038
loan_int_rate          0.000000
person_emp_exp         0.000000
person_home_ownership_OWN 0.000000
person_home_ownership_OTHER 0.000000
loan_intent_EDUCATION   0.000000
person_gender_None      0.000000
person_education_Master 0.000000
person_education_None    0.000000
person_education_Bachelor 0.000000
person_education_Doctorate 0.000000
loan_intent_HOMEIMPROVEMENT 0.000000
loan_intent_PERSONAL    0.000000
loan_intent_VENTURE      0.000000
person_home_ownership_None 0.000000
loan_intent_None         0.000000
credit_score            -0.000076
person_age               -0.000115
person_income             -0.008593
dtype: float64
AUC: 0.8222

```





Dal 50% al 70% ci sono stati peggioramenti delle performance del decision tree e del MLP sebbene a velocità diverse.

Nel Naive-Bayes, invece, si può notare un andamento “ondulatorio” delle performance ed in particolare della capacità di classificazione visualizzabile nella confusion matrix.

Osservando le features utilizzate è bene notare come più alti sono i valori di ‘person_income’ e ‘loan_int_rate’ maggiore è il miglioramento della abilità di classificare correttamente i valori.

Questo è probabilmente causato dal fatto che, eliminando sempre più valori, naive-bayes ha aumentato l’importanza di queste due features che, incidentalmente, descrivono meglio il modello.

Con valori superiori al 70% i vari modelli soffrono di severo underfitting a causa dell’eliminazione dei dati.

Inserimento valori NaN 25%,45%

Di seguito analizzeremo come l'impatto dell'inserimento ed imputazione dei valori nan ha influenzato il dataset durante la sporcatura precedente, in particolare in corrispondenza dei picchi "anomali".

I picchi sono presenti al 40% ed al 70% di sporcatura solo per l'algoritmo Naive-Bayes.

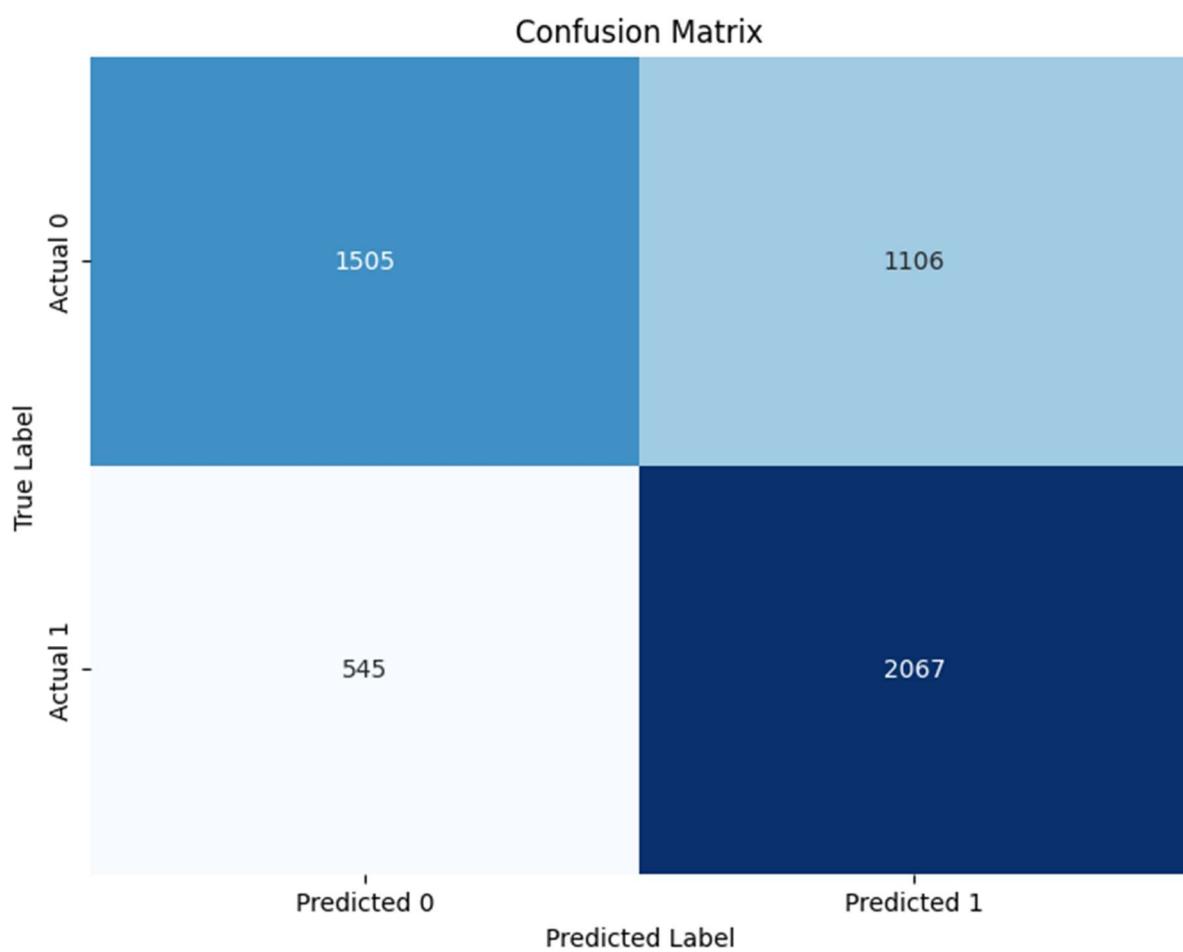
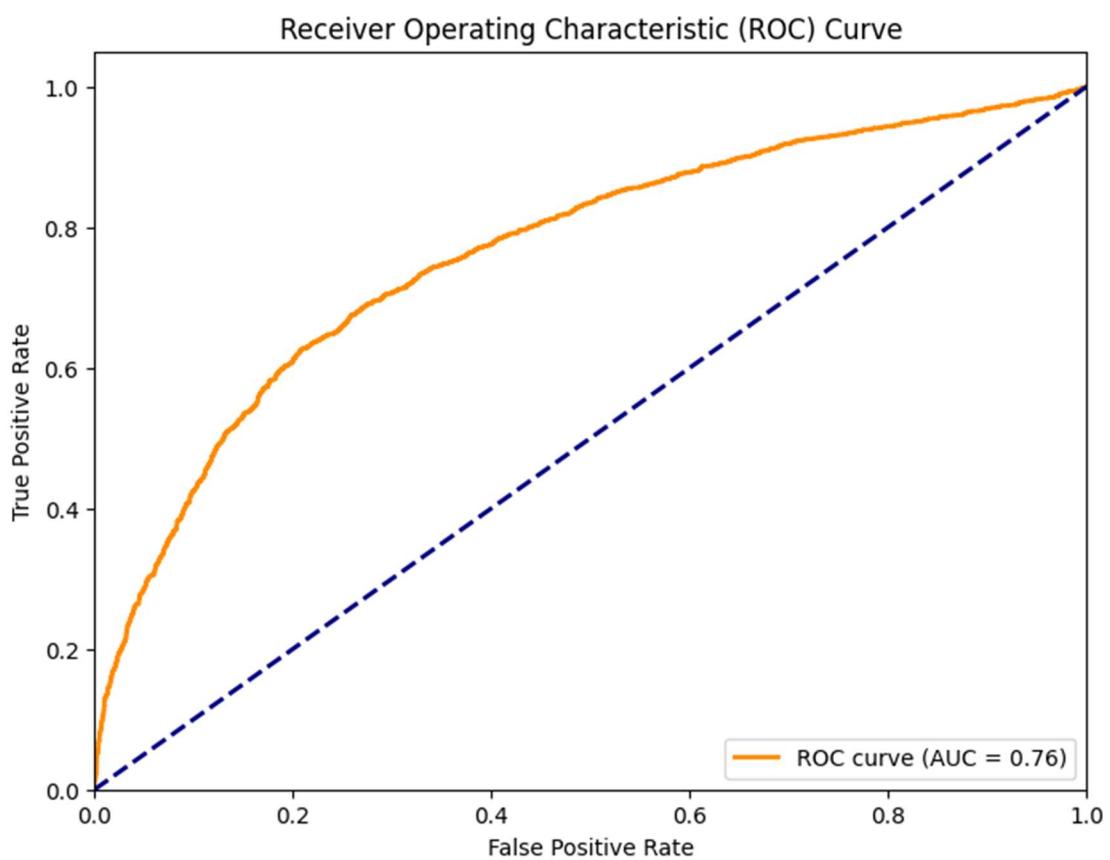
Considerando che il training set ha 134'409 celle sporcabili dal precedente algoritmo, utilizzando i valori contatore stampati nell'algoritmo possiamo calcolare che al picco 40%, circa 25% dei valori del dataset sporcati erano numerici, mentre al 70% di sporcatura erano 45%.

Quindi analizzeremo solo 25% e 45% del Naive-bayes.

25% di sporcatura:

```
Model: GaussianNB
Accuracy test: 0.6839 | Accuracy Training: 0.6518
F1-score test: 0.7146 | F1-score Training: 0.6945
roc_AUC-score test: 0.7636 | roc_AUC-score Training: 0.7305
Differenza Accuracy: -0.0321
Differenza F1-score: -0.0201
Differenza AUC: -0.0330

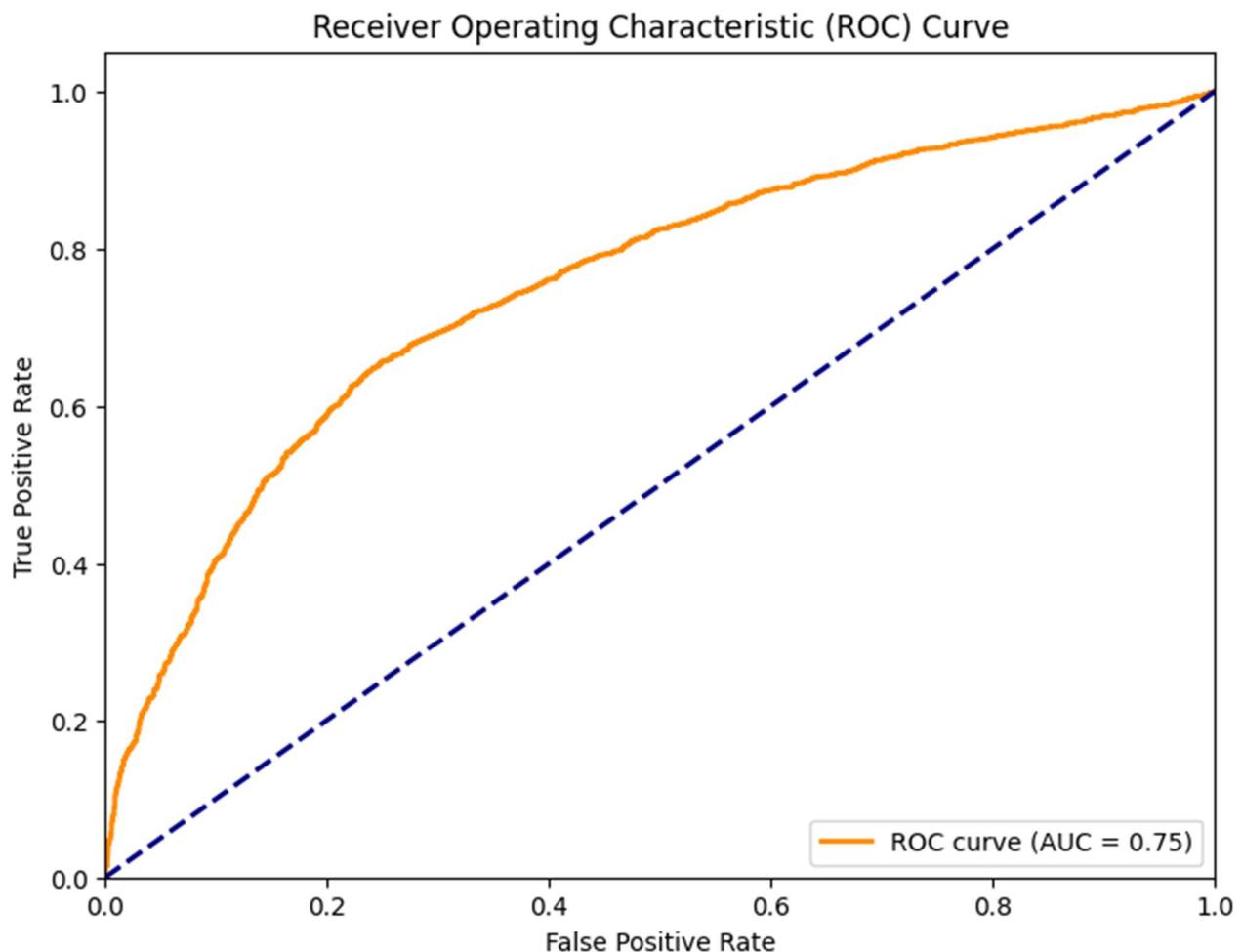
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'loan_int_rate': 0.0811
'person_income': 0.0412
'credit_score': 0.0286
'loan_amnt': 0.0200
'person_home_ownership_RENT': 0.0032
'loan_intent_VENTURE': 9.5730e-04
'person_home_ownership_OWN': 4.5951e-04
'loan_intent_MEDICAL': 1.5317e-04
'person_gender_male': 1.1488e-04
'loan_intent_HOMEIMPROVEMENT': 1.1488e-04
'person_education_Bachelor': 7.6584e-05
'loan_intent_EDUCATION': 4.4409e-17
'person_education_Master': -1.5317e-04
'loan_intent_PERSONAL': -2.2975e-04
'person_education_High School': -6.1267e-04
'cb_person_cred_hist_length': -0.0020
'person_emp_exp': -0.0044
'person_age': -0.0050
AUC: 0.7636
```

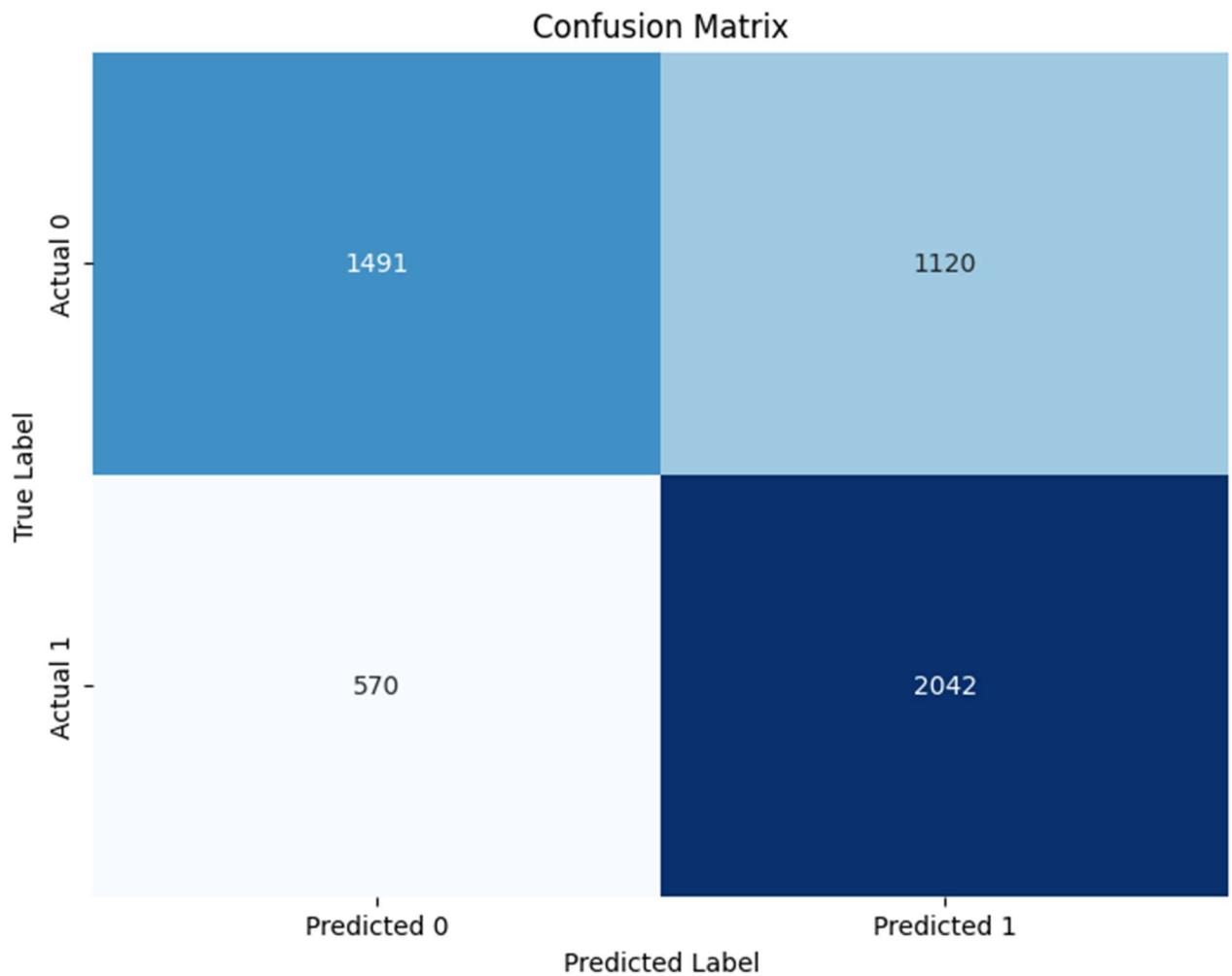


45% di sporcatura:

```
Model: GaussianNB
Accuracy test: 0.6764 | Accuracy Training: 0.6342
F1-score test: 0.7073 | F1-score Training: 0.6797
roc_AUC-score test: 0.7525 | roc_AUC-score Training: 0.7015
Differenza Accuracy: -0.0422
Differenza F1-score: -0.0276
Differenza AUC: -0.0510

Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'loan_int_rate': 0.0811
'person_income': 0.0319
'credit_score': 0.0303
'loan_amnt': 0.0184
'person_home_ownership_RENT': 0.0060
'loan_intent_VENTURE': 0.0011
'person_home_ownership_OWN': 4.2121e-04
'person_education_Bachelor': 2.6805e-04
'loan_intent_HOMEIMPROVEMENT': 1.5317e-04
'loan_intent_MEDICAL': 1.1488e-04
'person_education_High School': 7.6584e-05
'person_education_Master': 3.8292e-05
'loan_intent_PERSONAL': -2.2975e-04
'loan_intent_EDUCATION': -3.4463e-04
'cb_person_cred_hist_length': -0.0034
'person_emp_exp': -0.0040
'person_age': -0.0051
AUC: 0.7525
```





Da notare come il livello di confusione nelle matrici è comparabile a quello identificato precedentemente a livello 40% e 70% di sporcatura.

Inoltre, è bene notare come pure in questi casi ‘loan_int_rate’ e ‘person_income’ hanno un relativamente alto livello di importanza nell’addestramento.

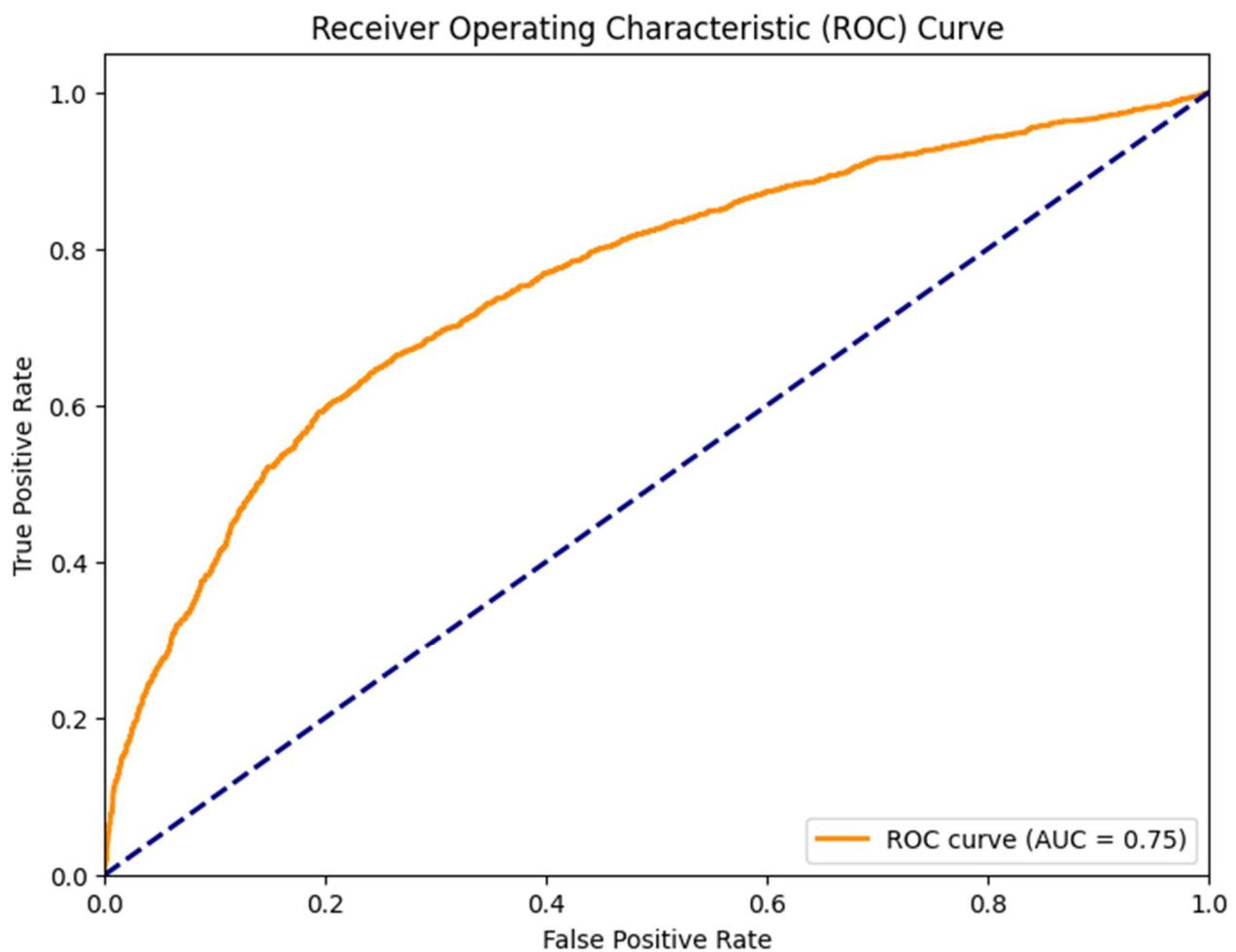
Questa combinazione di fattori ci permette di dire che sebbene l’imputazione dei valori NaN possa aver avuto qualche effetto imprevisto dei dati, non è possibile escludere che le features ‘loan_int_rate’ e ‘person_income’ siano semplicemente molto efficaci per la modellazione del sistema predittivo e, incidentalmente, quando il 25% ed il 45% dei valori numerici sono NaN (e poi imputati) quelle due features finiscono per risultare più importanti per il modello di Naive-Bayes.

Ripetizioni di voci nel dataset 30%-80%-999%

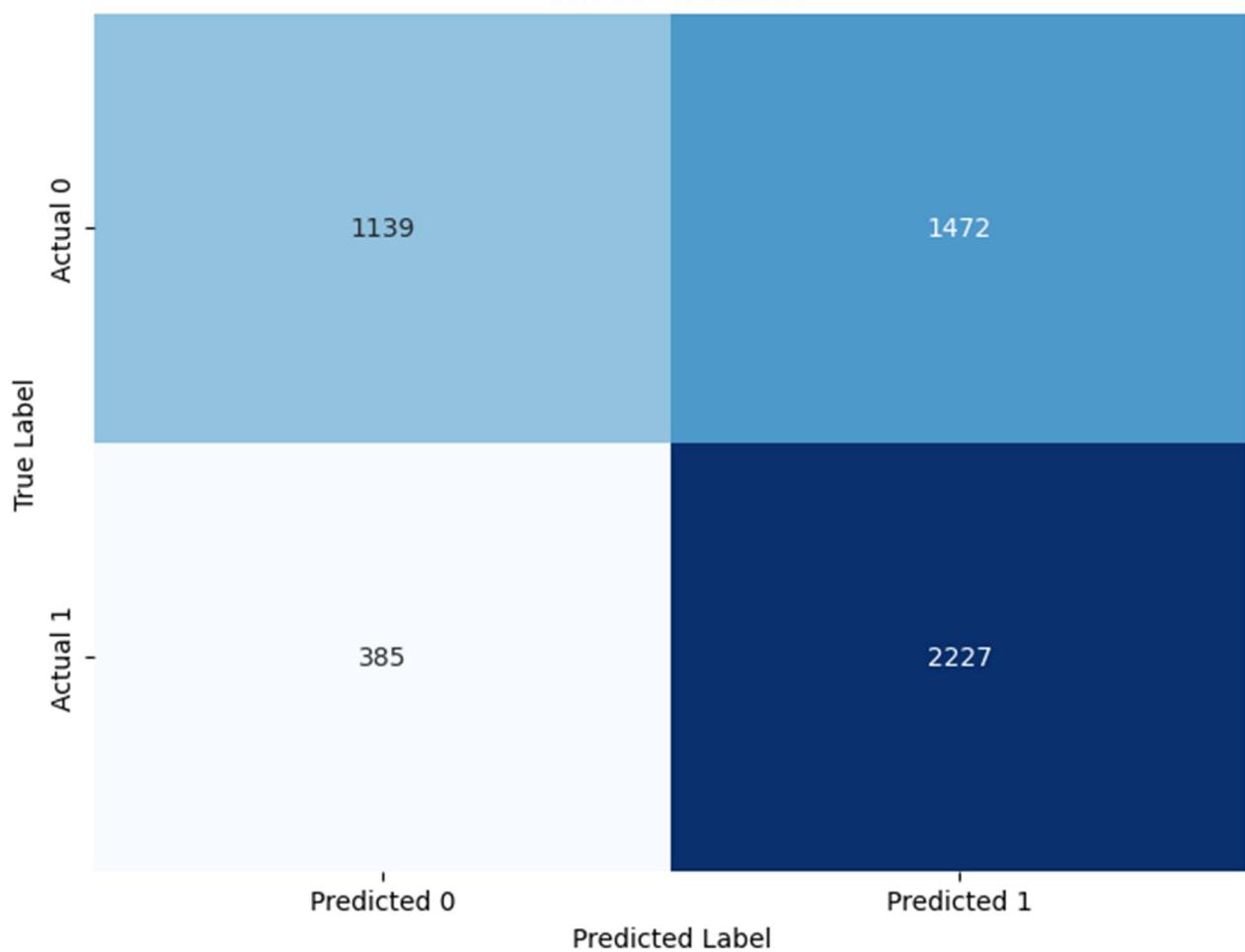
30% di sporcatura:

```
Aggiunti 3656 duplicati. Dimensione dataset finale: 12187
Model: GaussianNB
Accuracy test: 0.6445 | Accuracy Training: 0.6379
F1-score test: 0.7058 | F1-score Training: 0.7032
roc_AUC-score test: 0.7543 | roc_AUC-score Training: 0.7452
Differenza Accuracy: -0.0066
Differenza F1-score: -0.0025
Differenza AUC: -0.0091

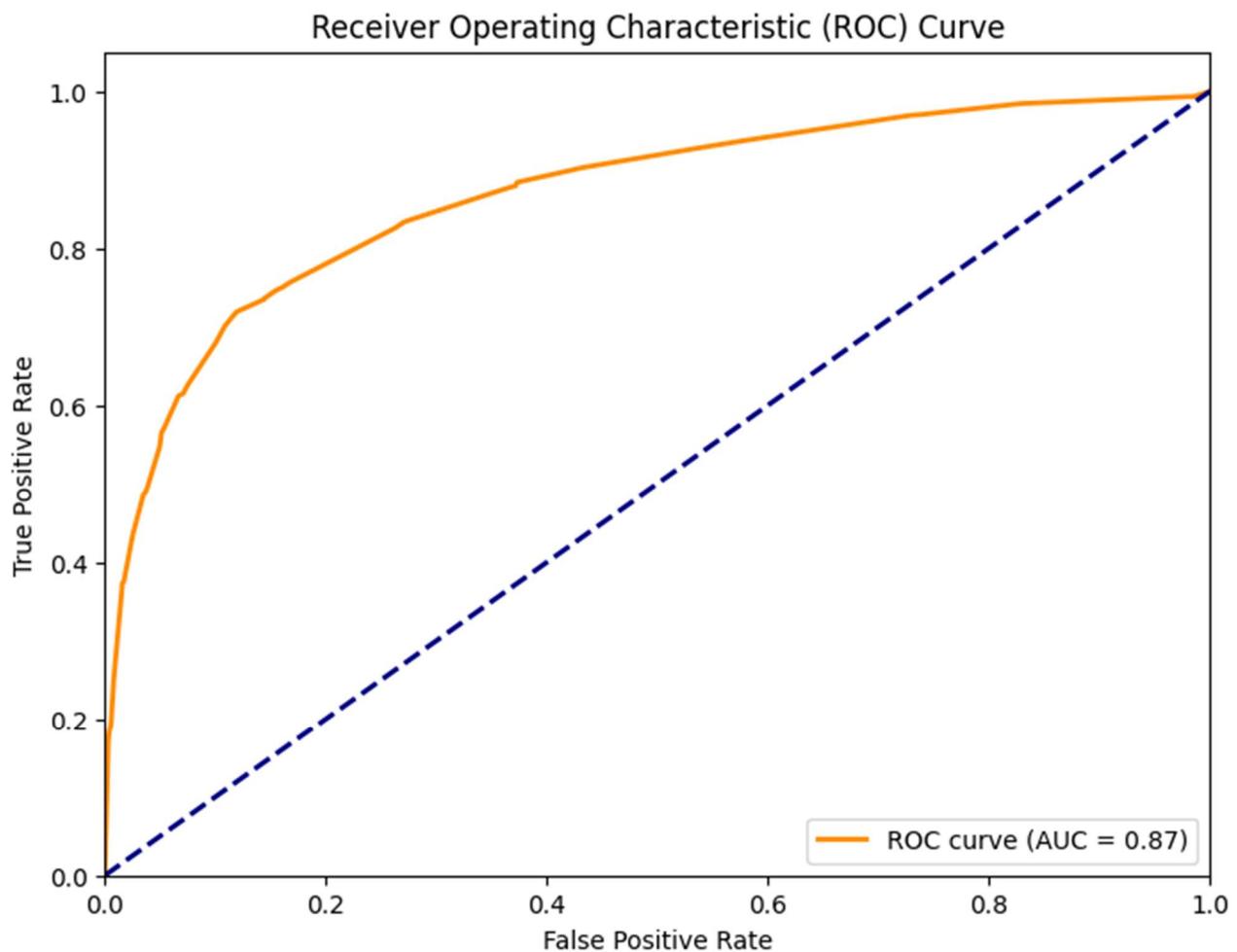
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'loan_int_rate': 0.0627
'person_income': 0.0365
'credit_score': 0.0185
'loan_amnt': 0.0132
'person_home_ownership_RENT': 0.0015
'loan_intent_MEDICAL': 3.0634e-04
'person_education_Master': 1.9146e-04
'person_home_ownership_OWN': 1.9146e-04
'loan_intent_EDUCATION': 3.8292e-05
'loan_intent_PERSONAL': -2.2204e-17
'person_education_High School': -3.8292e-05
'loan_intent_VENTURE': -2.6805e-04
'cb_person_cred_hist_length': -0.0044
'person_age': -0.0078
'person_emp_exp': -0.0081
AUC: 0.7543
```



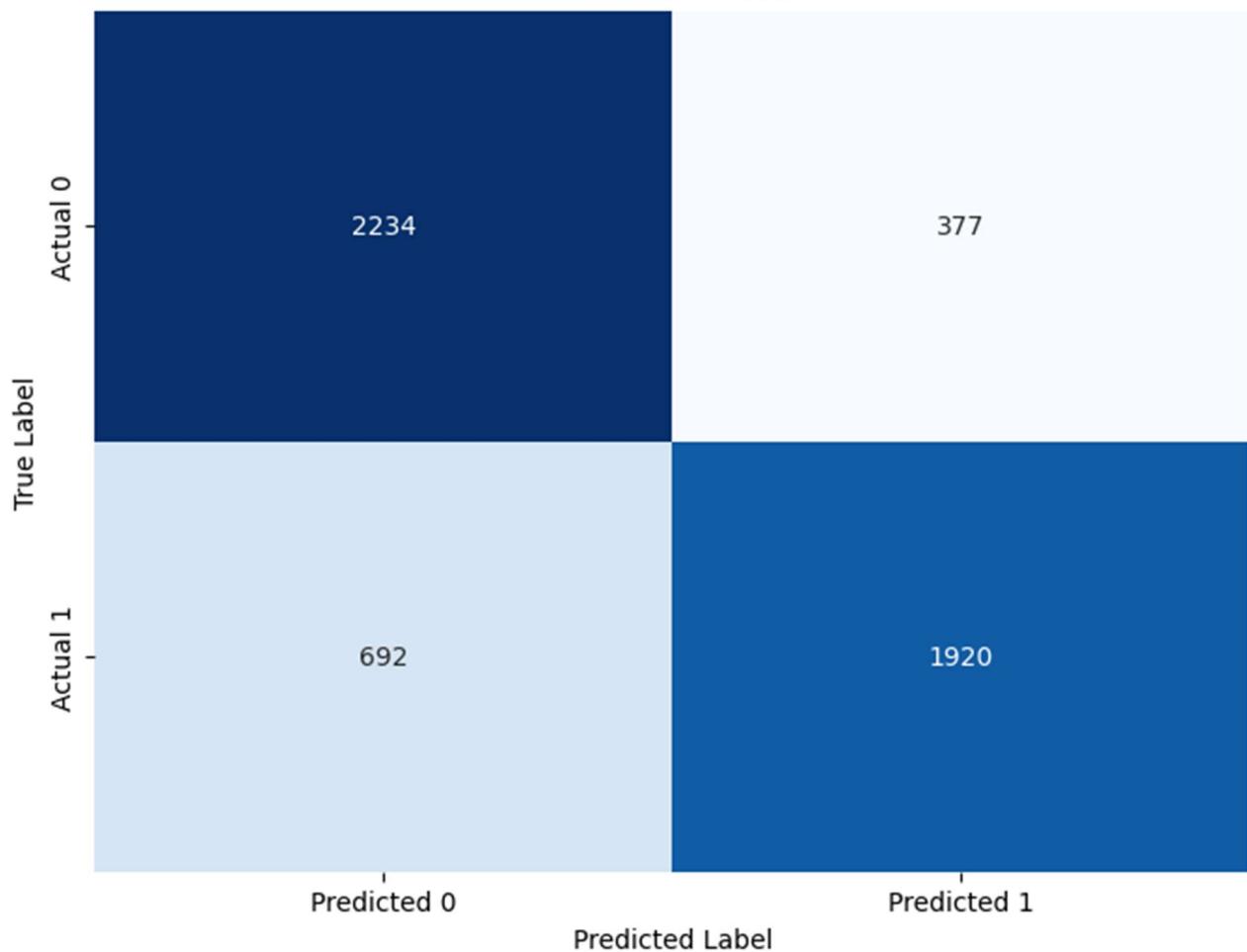
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.7953 | Accuracy Training: 0.8246
F1-score test: 0.7822 | F1-score Training: 0.8125
roc_AUC-score test: 0.8674 | roc_AUC-score Training: 0.8968
Differenza Accuracy: 0.0293
Differenza F1-score: 0.0303
Differenza AUC: 0.0294
Importanza delle Feature per Decision Tree:
'loan_int_rate': 0.3066
'person_income': 0.2994
'loan_amnt': 0.1673
'person_home_ownership_RENT': 0.1512
'credit_score': 0.0222
'loan_intent_HOMEIMPROVEMENT': 0.0189
'loan_intent_MEDICAL': 0.0128
'person_home_ownership_OWN': 0.0106
'loan_intent_VENTURE': 0.0061
'person_age': 0.0027
'loan_intent_PERSONAL': 0.0014
'person_emp_exp': 5.6510e-04
'cb_person_cred_hist_length': 1.4507e-04
AUC: 0.8674
```

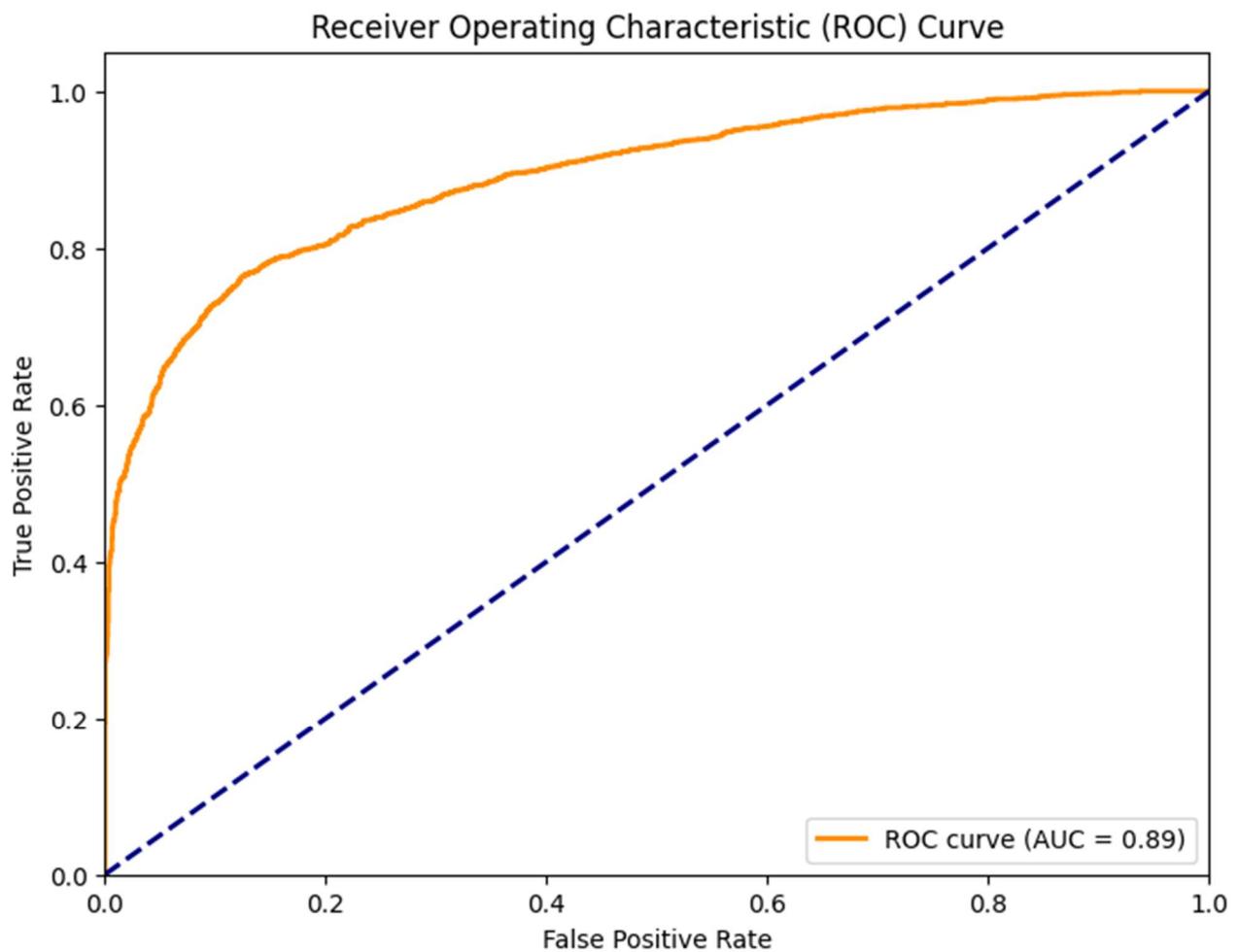


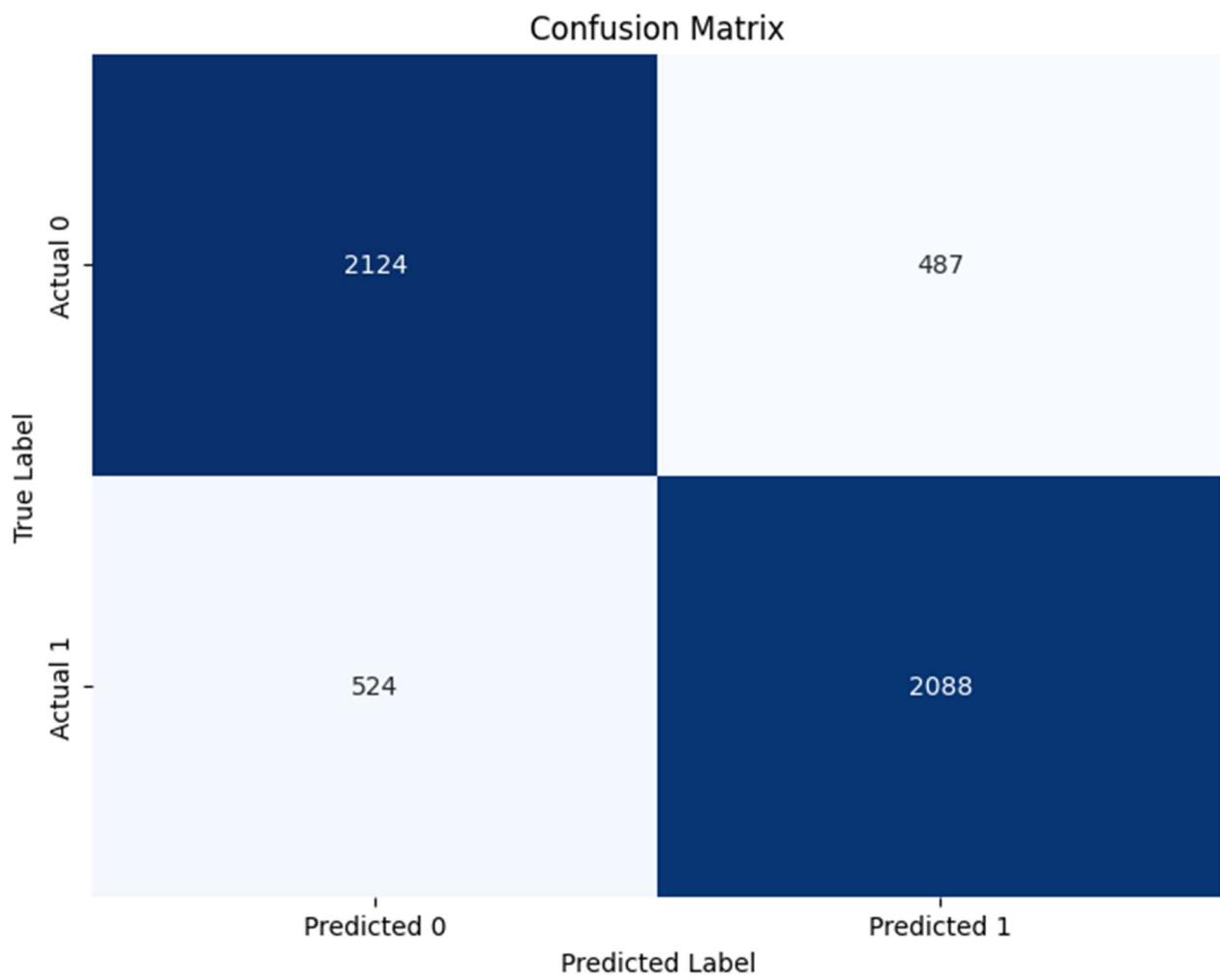
Confusion Matrix



```
Model: MLPClassifier
Accuracy test: 0.8064 | Accuracy Training: 0.8699
F1-score test: 0.8051 | F1-score Training: 0.8670
roc_AUC-score test: 0.8905 | roc_AUC-score Training: 0.9431
Differenza Accuracy: 0.0634
Differenza F1-score: 0.0619
Differenza AUC: 0.0526

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age          0.000000
person_emp_exp       0.000000
loan_amnt            0.000000
loan_int_rate         0.000000
person_home_ownership_OTHER 0.000000
cb_person_cred_hist_length 0.000000
credit_score          0.000000
person_gender_male    0.000000
person_education_Bachelor 0.000000
person_education_Doctorate 0.000000
person_education_High School 0.000000
person_education_Master 0.000000
loan_intent_HOMEIMPROVEMENT 0.000000
person_home_ownership_OWN 0.000000
person_home_ownership_RENT 0.000000
loan_intent_EDUCATION 0.000000
loan_intent_PERSONAL 0.000000
loan_intent_MEDICAL 0.000000
loan_intent_VENTURE 0.000000
person_income          -0.000038
dtype: float64
AUC: 0.8905
```





Da notare il netto miglioramento nella matrice di confusione del Naive-Bayes rispetto ai valori originali.

I valori continuano a migliorare stabilmente per il naive-bayes e rimangono costanti fino all' 80% di sporcatura.

La tendenza al miglioramento del Naive-Bayes è probabilmente causata dal fatto che questo tipo di sporcatura rafforza le credenze del modello introducendo delle ridondanze.

Per iniziare a vedere dei netti peggioramenti nei valori statistici e grafici si è dovuto arrivare al 999% di sporcatura.

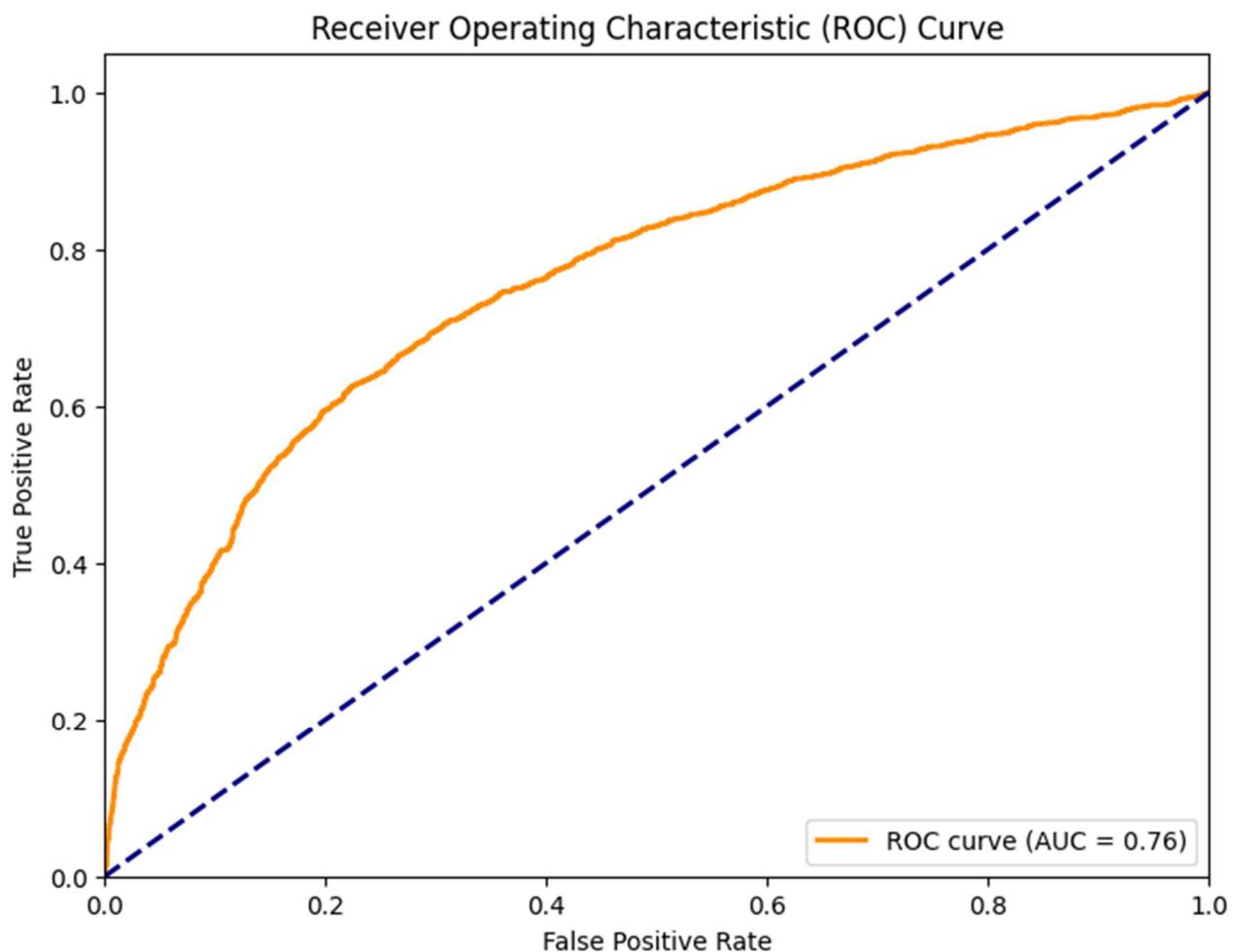
Al 999% di sporcatura si inizia a notare l'overfitting del modello.

Di seguito mostro i valori alle percentuali sopracitati

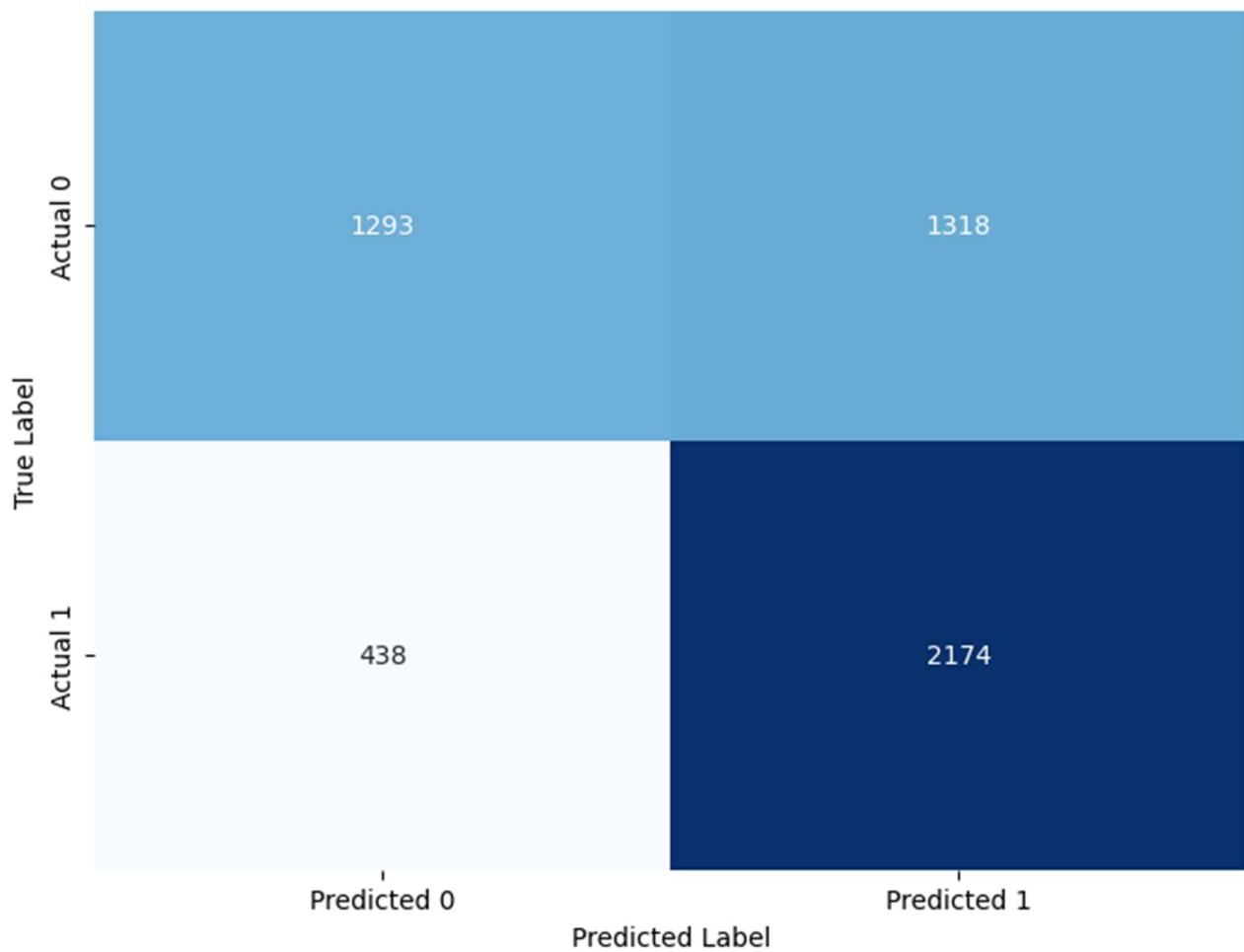
80% di sporcatura:

```
Aggiunti 9749 duplicati. Dimensione dataset finale: 12187
Model: GaussianNB
Accuracy test: 0.6638 | Accuracy Training: 0.6641
F1-score test: 0.7123 | F1-score Training: 0.7089
roc_AUC-score test: 0.7564 | roc_AUC-score Training: 0.7549
Differenza Accuracy: 0.0003
Differenza F1-score: -0.0034
Differenza AUC: -0.0015

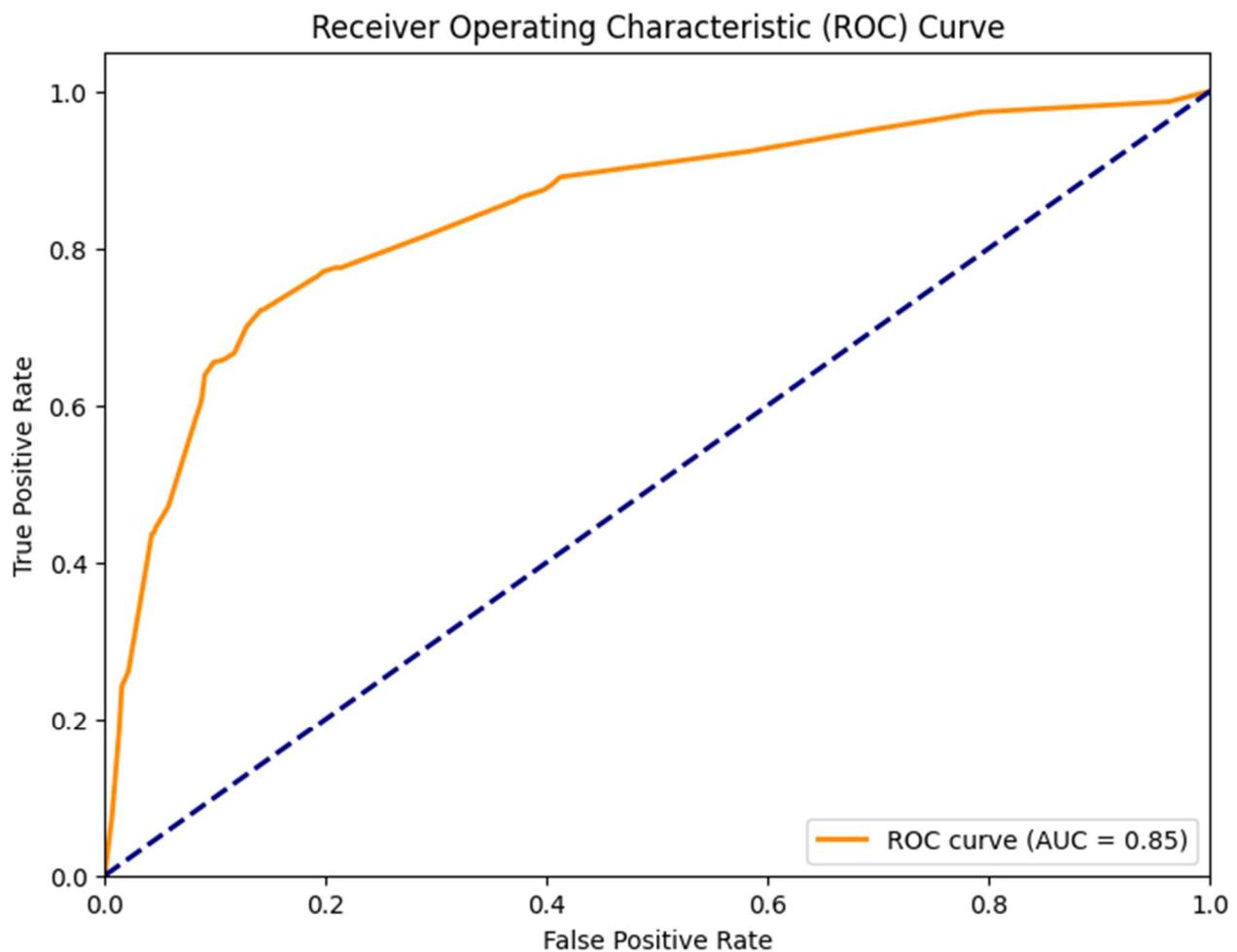
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'loan_int_rate': 0.0772
'person_income': 0.0329
'credit_score': 0.0264
'loan_amnt': 0.0194
'person_home_ownership_RENT': 0.0013
'loan_intent_MEDICAL': 4.9780e-04
'person_education_Bachelor': 3.8292e-04
'loan_intent_EDUCATION': 3.4463e-04
'loan_intent_VENTURE': 1.9146e-04
'person_gender_male': 1.1488e-04
'person_home_ownership_OWN': 3.8292e-05
'loan_intent_PERSONAL': -2.2204e-17
'person_education_Master': -1.9146e-04
'person_education_High School': -2.2975e-04
'cb_person_cred_hist_length': -0.0013
'person_emp_exp': -0.0023
'person_age': -0.0034
AUC: 0.7564
```



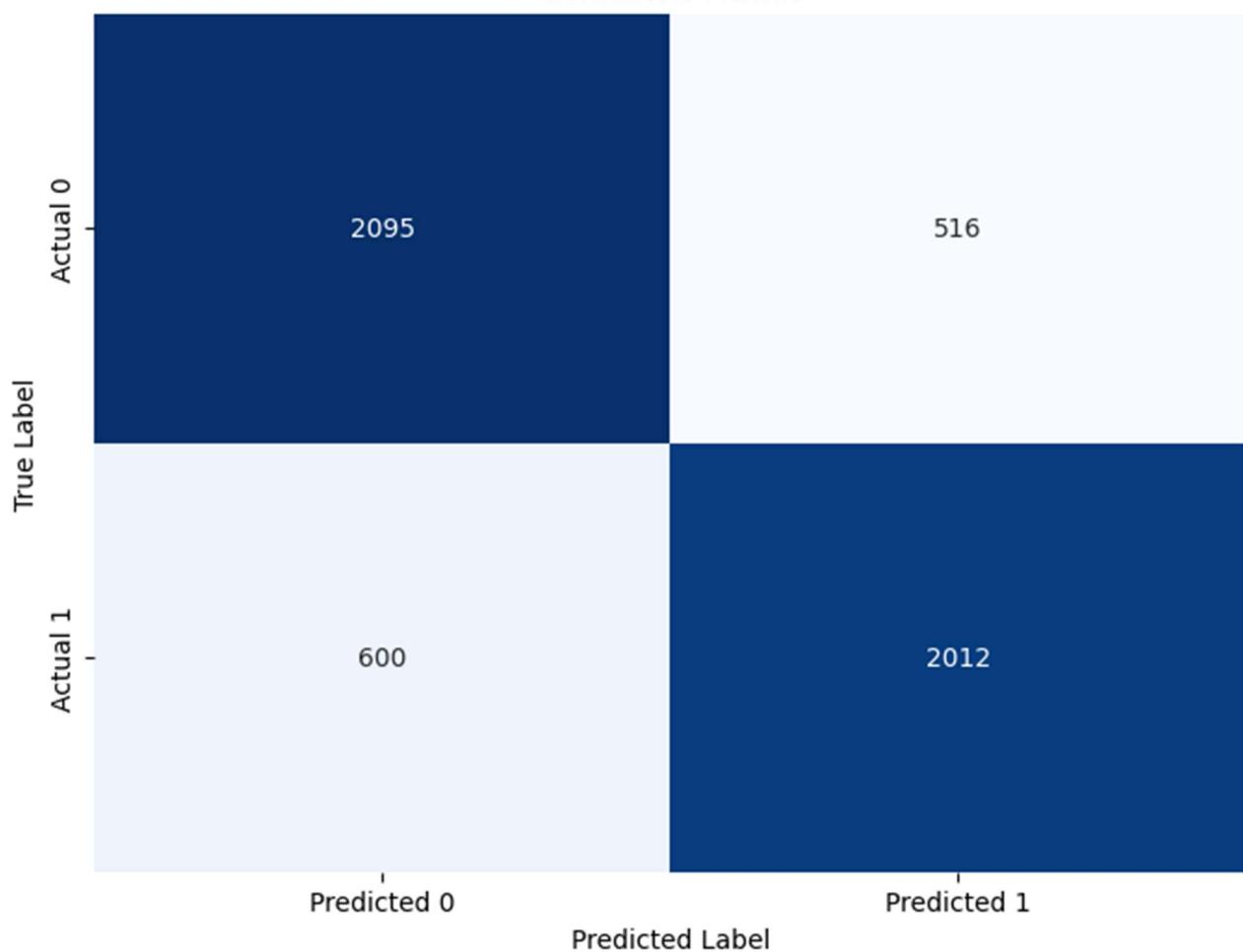
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.7863 | Accuracy Training: 0.8484
F1-score test: 0.7829 | F1-score Training: 0.8435
roc_AUC-score test: 0.8462 | roc_AUC-score Training: 0.9163
Differenza Accuracy: 0.0621
Differenza F1-score: 0.0607
Differenza AUC: 0.0702
Importanza delle Feature per Decision Tree:
'loan_int_rate': 0.3255
'person_income': 0.3045
'loan_amnt': 0.1676
'person_home_ownership_RENT': 0.1013
'person_home_ownership_OWN': 0.0314
'credit_score': 0.0258
'person_education_Bachelor': 0.0146
'loan_intent_PERSONAL': 0.0092
'cb_person_cred_hist_length': 0.0064
'loan_intent_VENTURE': 0.0051
'person_emp_exp': 0.0041
'loan_intent_MEDICAL': 0.0032
'person_age': 0.0013
AUC: 0.8462
```



Confusion Matrix

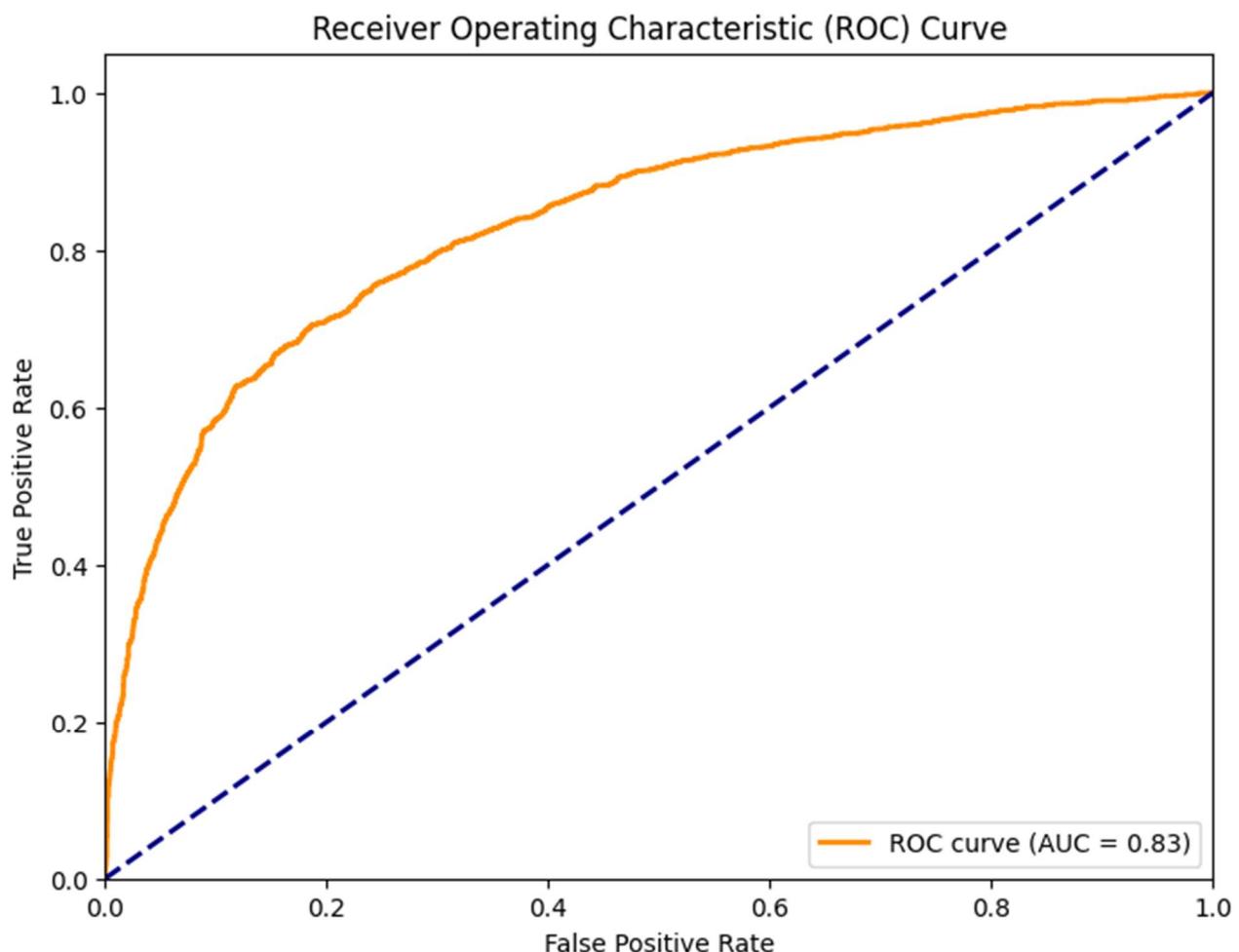


```

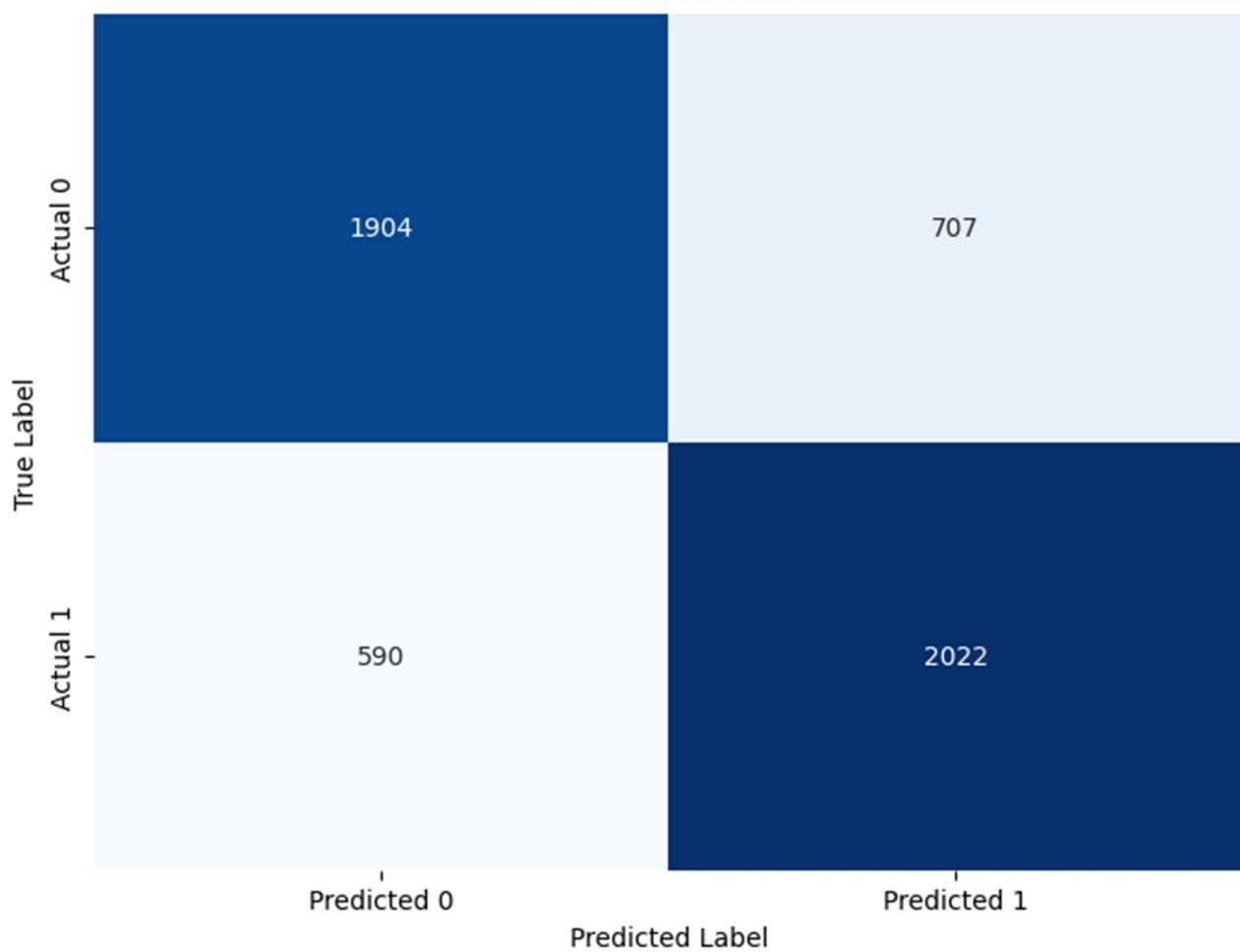
Model: MLPClassifier
Accuracy test: 0.7517 | Accuracy Training: 0.9520
F1-score test: 0.7572 | F1-score Training: 0.9518
roc_AUC-score test: 0.8331 | roc_AUC-score Training: 0.9894
Differenza Accuracy: 0.2003
Differenza F1-score: 0.1946
Differenza AUC: 0.1563

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age          0.000000
person_emp_exp       0.000000
loan_amnt            0.000000
loan_int_rate         0.000000
person_home_ownership_OTHER 0.000000
cb_person_cred_hist_length 0.000000
credit_score          0.000000
person_gender_male    0.000000
person_education_Bachelor 0.000000
person_education_Doctorate 0.000000
person_education_High School 0.000000
person_education_Master 0.000000
loan_intent_HOMEIMPROVEMENT 0.000000
person_home_ownership_OWN 0.000000
person_home_ownership_RENT 0.000000
loan_intent_EDUCATION 0.000000
loan_intent_PERSONAL 0.000000
loan_intent_MEDICAL 0.000000
loan_intent_VENTURE 0.000000
person_income          -0.000038
dtype: float64
AUC: 0.8331

```



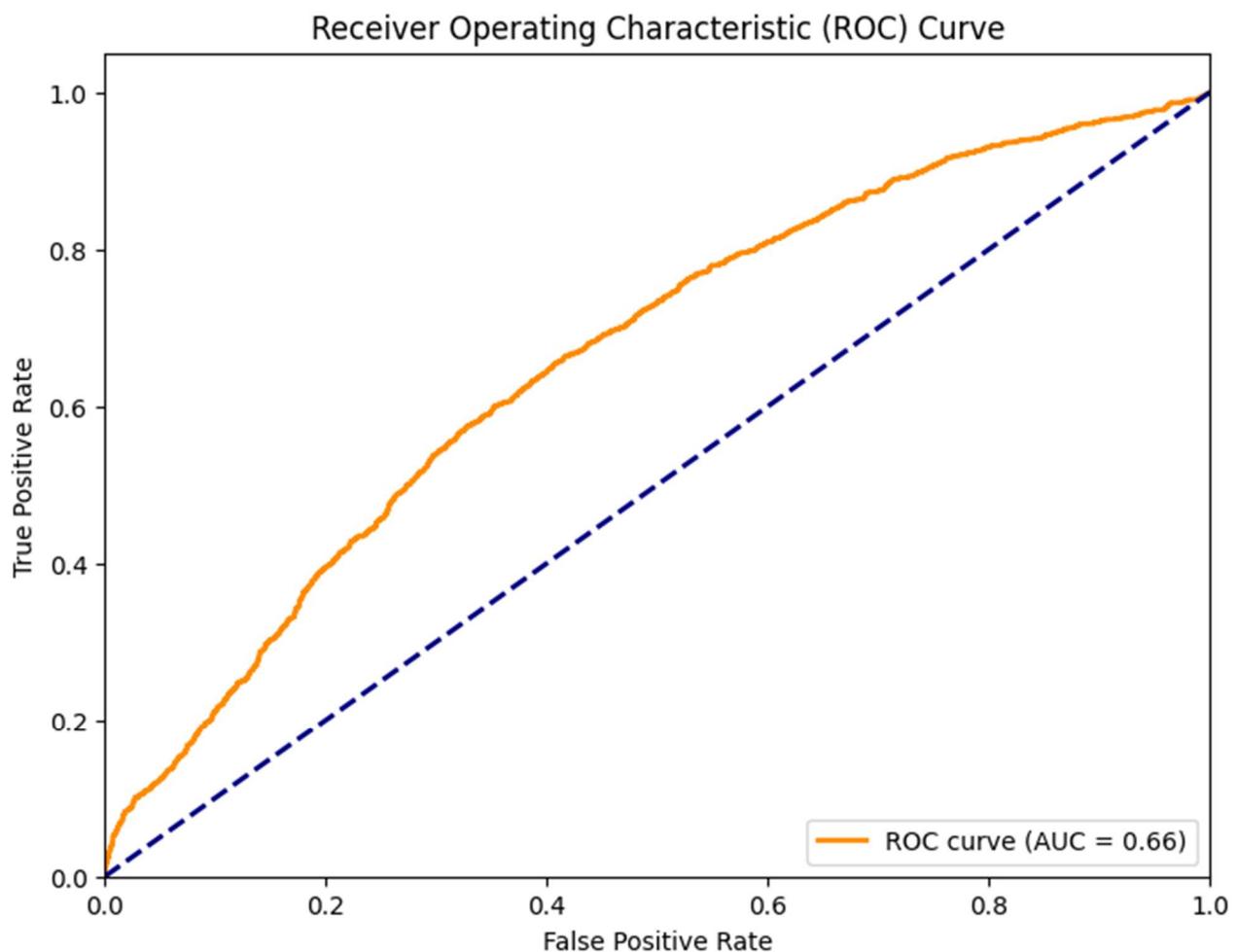
Confusion Matrix



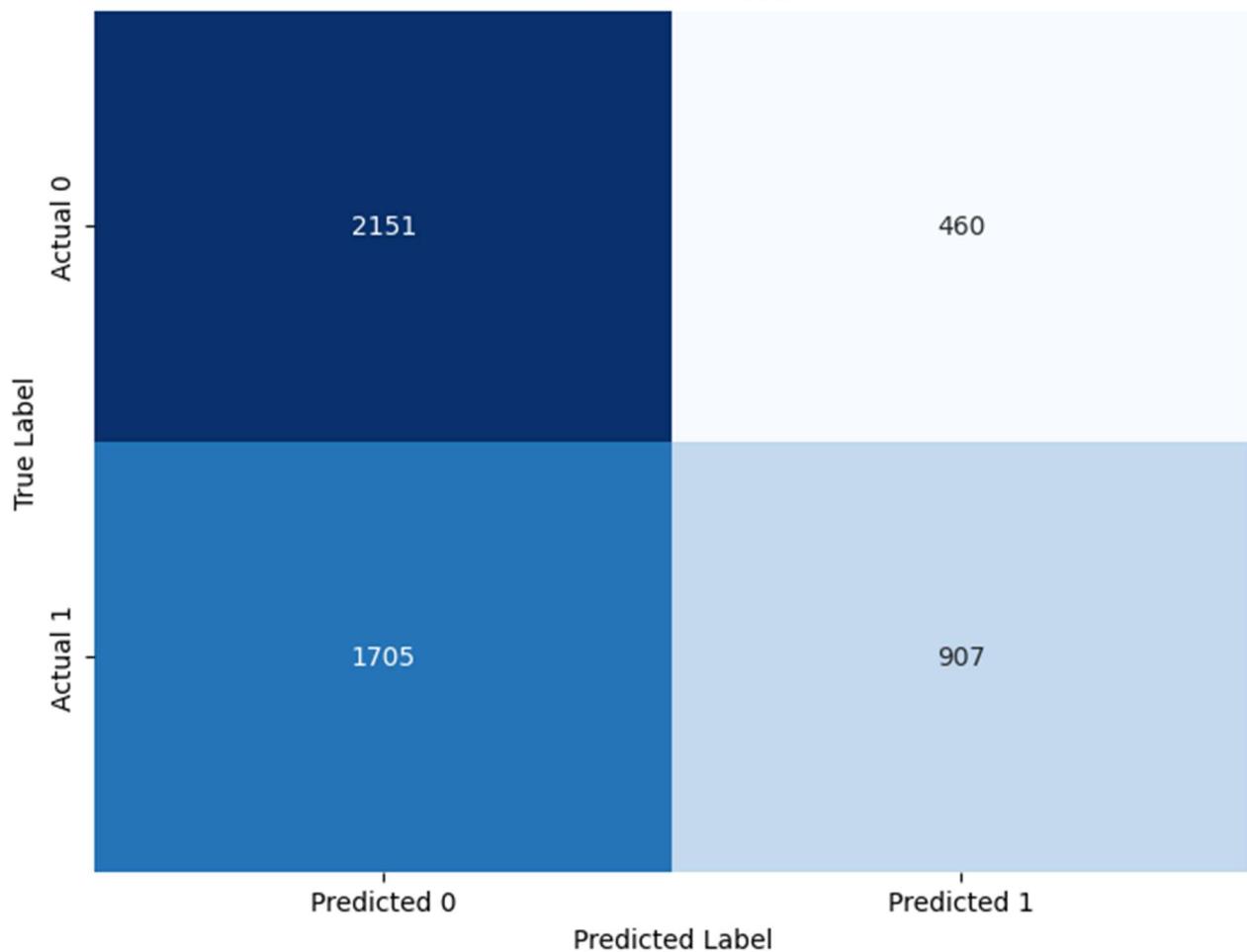
99% di sporcatura:

```
Model: GaussianNB
Accuracy test: 0.5855 | Accuracy Training: 0.8461
F1-score test: 0.4559 | F1-score Training: 0.8307
roc_AUC-score test: 0.6617 | roc_AUC-score Training: 1.0000
Differenza Accuracy: 0.2606
Differenza F1-score: 0.3749
Differenza AUC: 0.3383

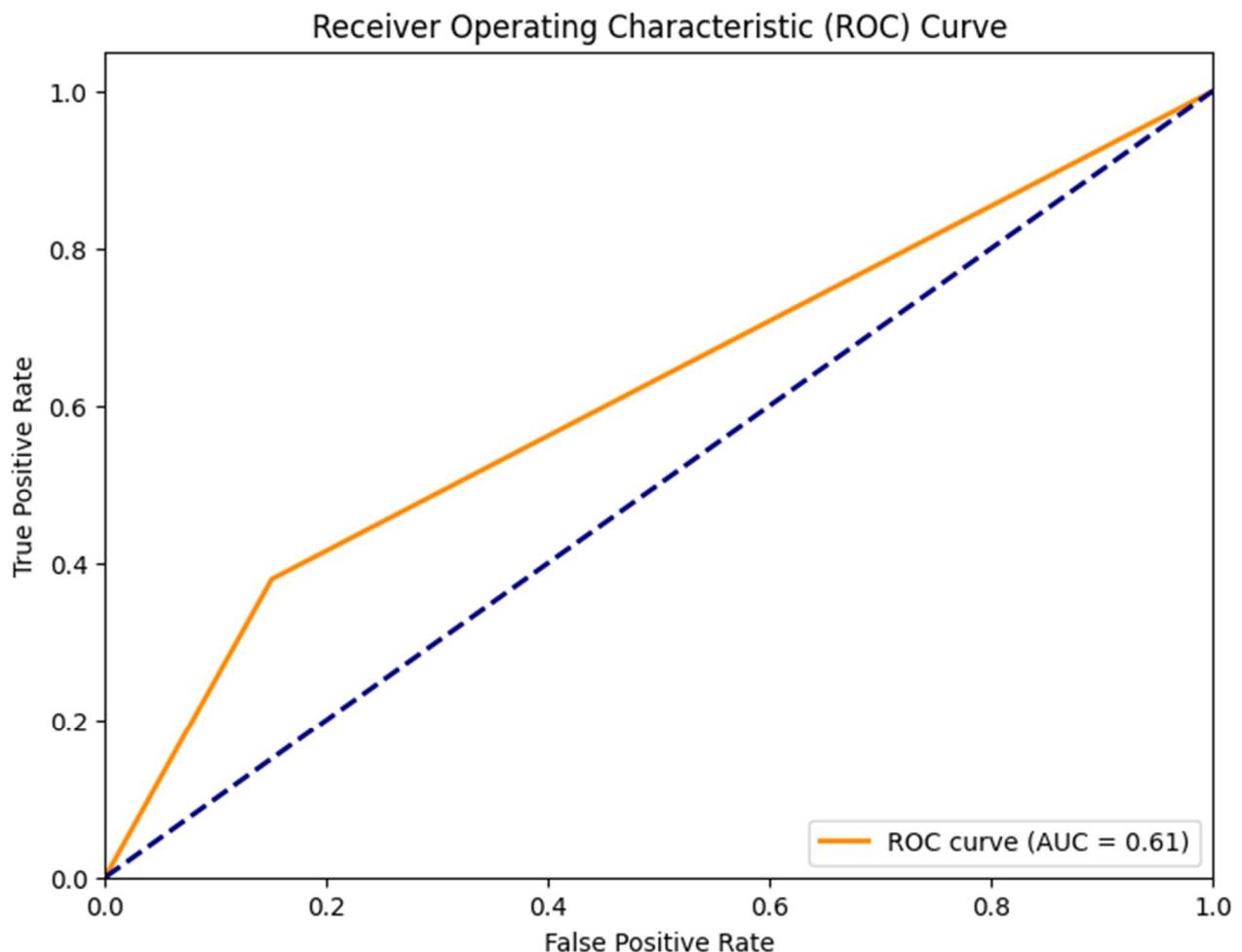
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'person_income': 0.0825
'person_age': 0.0103
'credit_score': 0.0101
'cb_person_cred_hist_length': 0.0082
'person_emp_exp': 0.0062
'loan_int_rate': 0.0026
'person_education_High School': 7.6584e-05
'person_home_ownership_RENT': 7.6584e-05
'loan_intent_MEDICAL': 3.8292e-05
'person_gender_male': -3.8292e-05
'loan_intent_PERSONAL': -7.6584e-05
'loan_intent_EDUCATION': -7.6584e-05
'loan_intent_VENTURE': -3.4463e-04
'loan_amnt': -0.0015
AUC: 0.6617
```



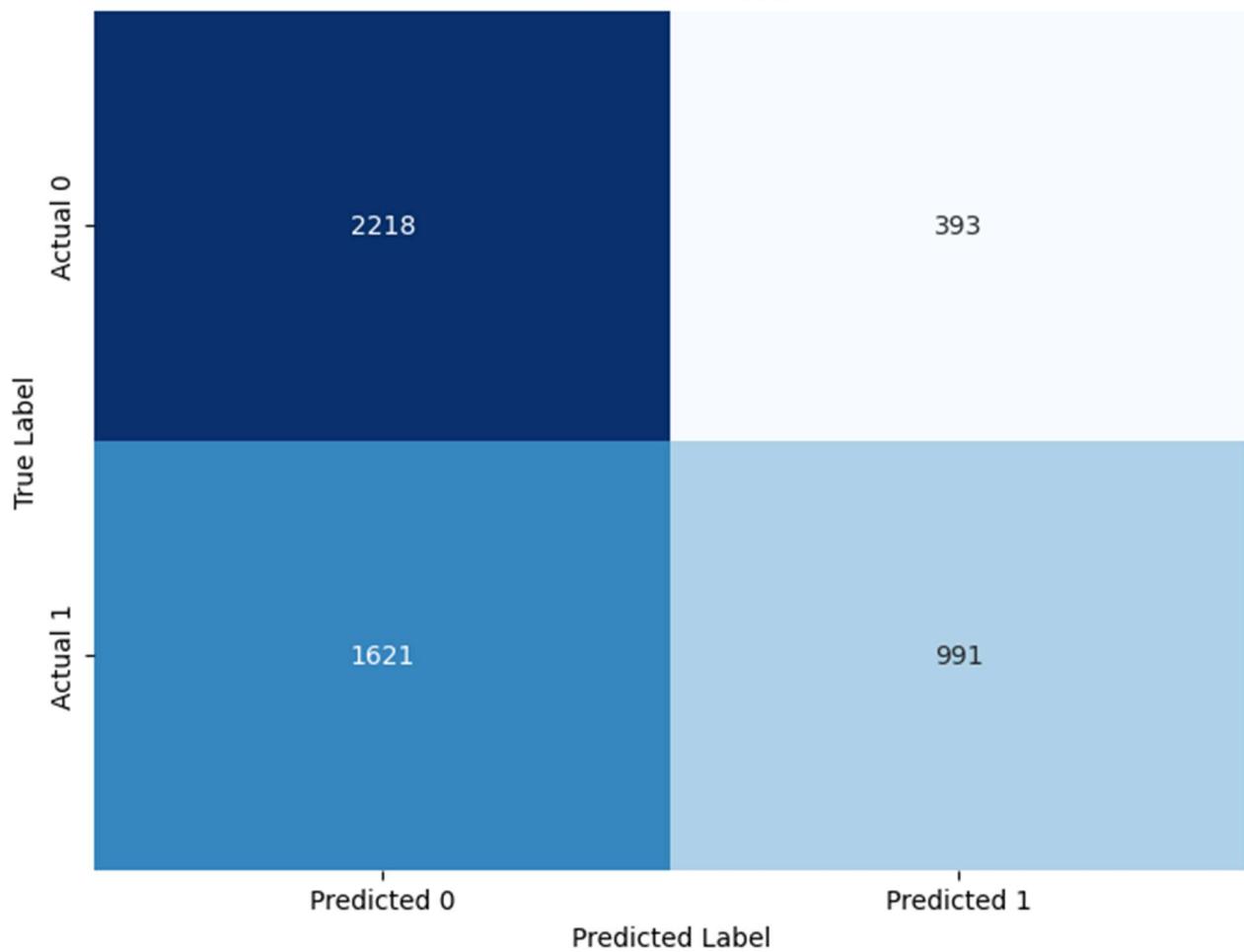
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.6144 | Accuracy Training: 1.0000
F1-score test: 0.4960 | F1-score Training: 1.0000
roc_AUC-score test: 0.6144 | roc_AUC-score Training: 1.0000
Differenza Accuracy: 0.3856
Differenza F1-score: 0.5040
Differenza AUC: 0.3856
Importanza delle Feature per Decision Tree:
'person_income': 0.7377
'loan_intent_EDUCATION': 0.2623
AUC: 0.6144
```

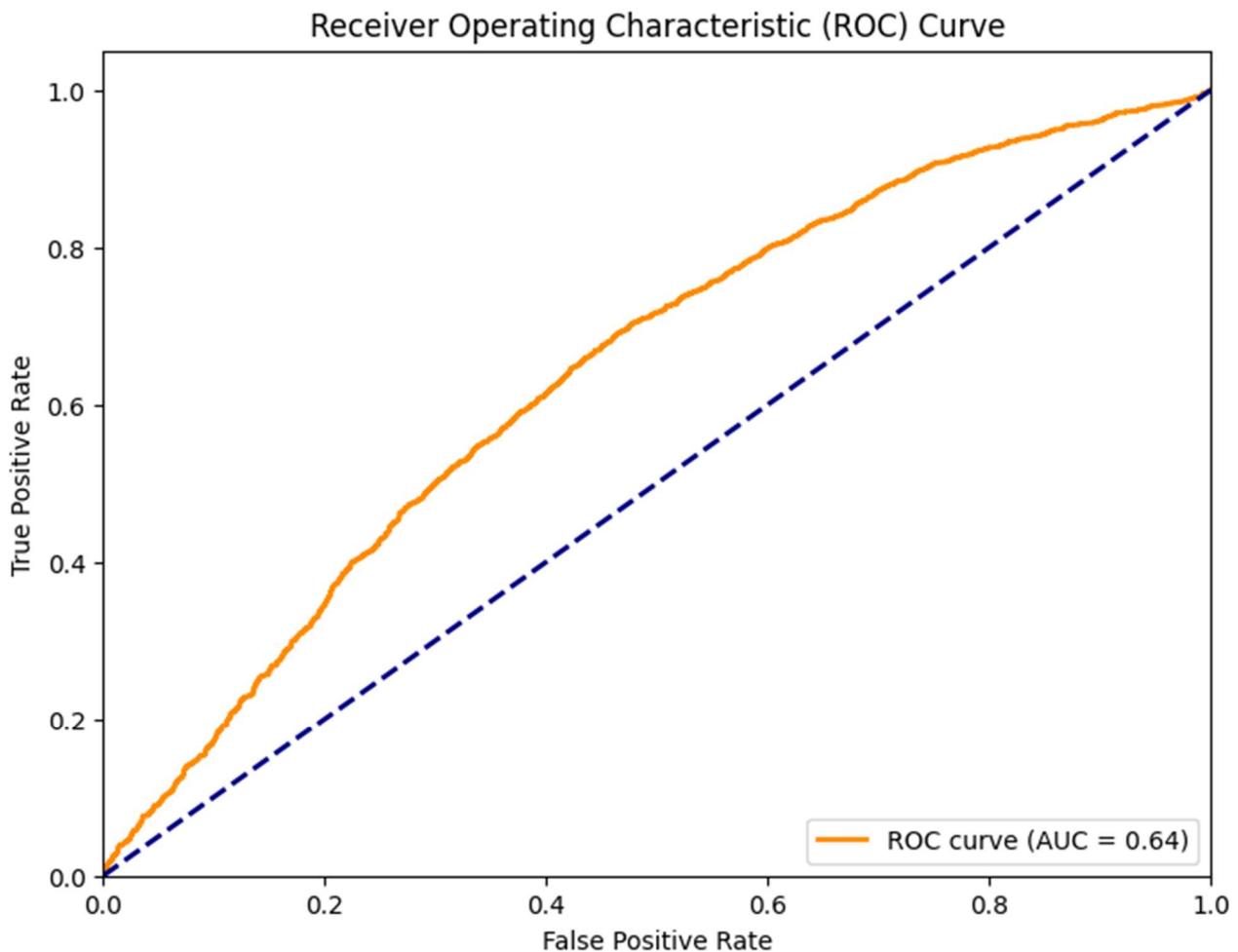


Confusion Matrix

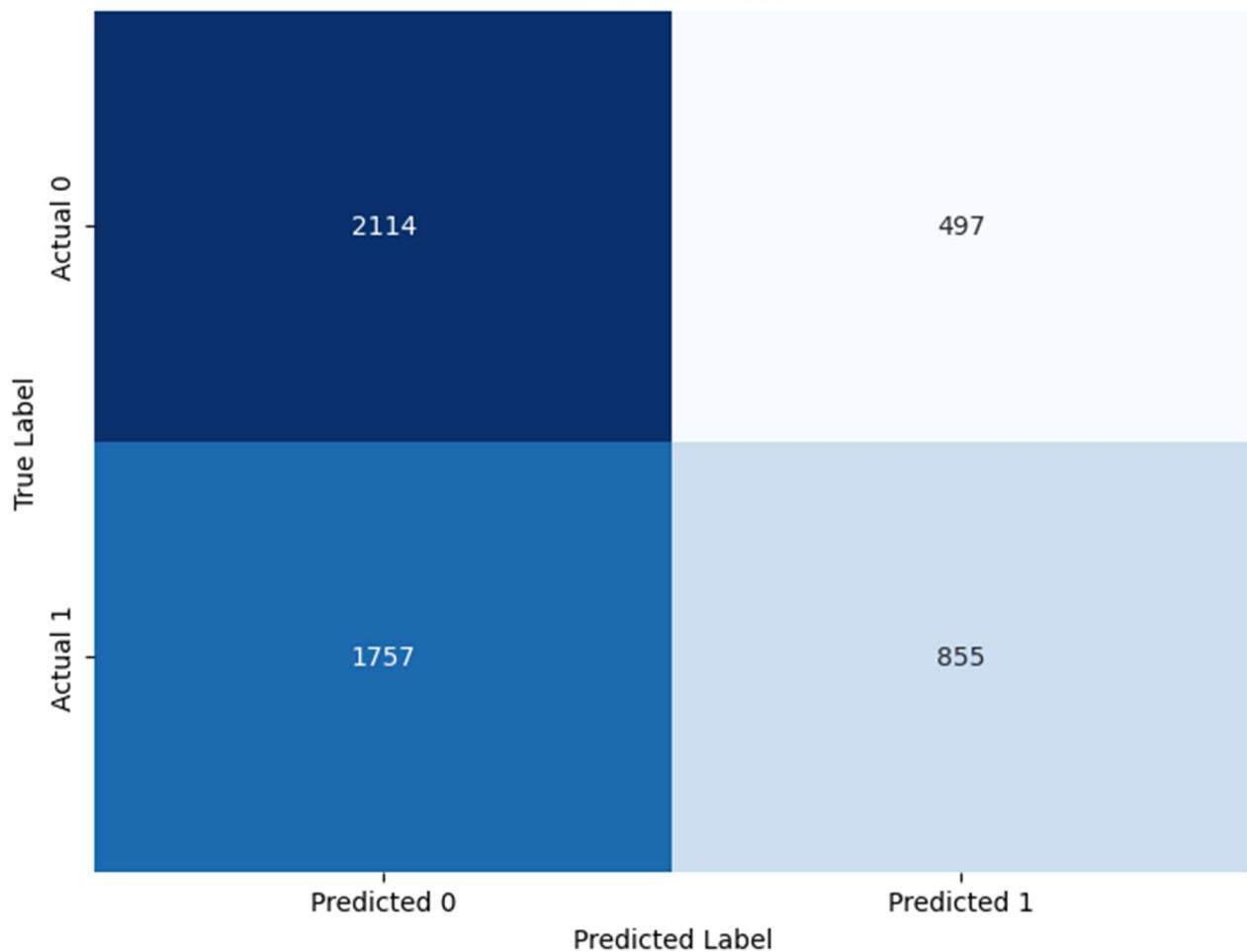


```
Model: MLPClassifier
Accuracy test: 0.5684 | Accuracy Training: 1.0000
F1-score test: 0.4314 | F1-score Training: 1.0000
roc_AUC-score test: 0.6426 | roc_AUC-score Training: 1.0000
Differenza Accuracy: 0.4316
Differenza F1-score: 0.5686
Differenza AUC: 0.3574

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age          0.0
person_income        0.0
person_emp_exp       0.0
loan_amnt           0.0
loan_int_rate        0.0
cb_person_cred_hist_length 0.0
credit_score         0.0
person_gender_male   0.0
person_education_Bachelor 0.0
person_education_High School 0.0
person_education_Master 0.0
person_home_ownership_RENT 0.0
loan_intent_EDUCATION 0.0
loan_intent_MEDICAL 0.0
loan_intent_PERSONAL 0.0
loan_intent_VENTURE 0.0
dtype: float64
AUC: 0.6426
```



Confusion Matrix

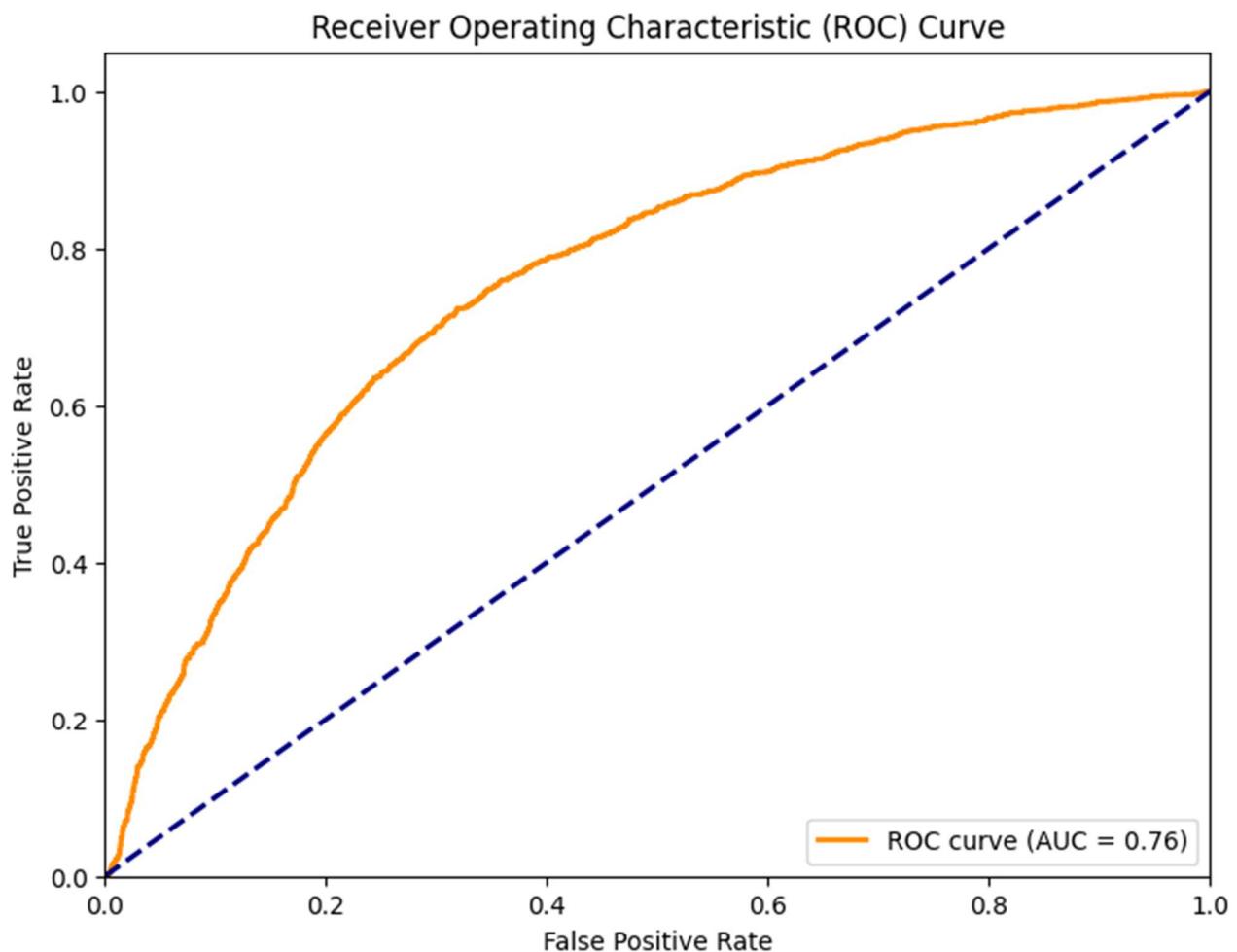


Errori di Selezione 20%-40%

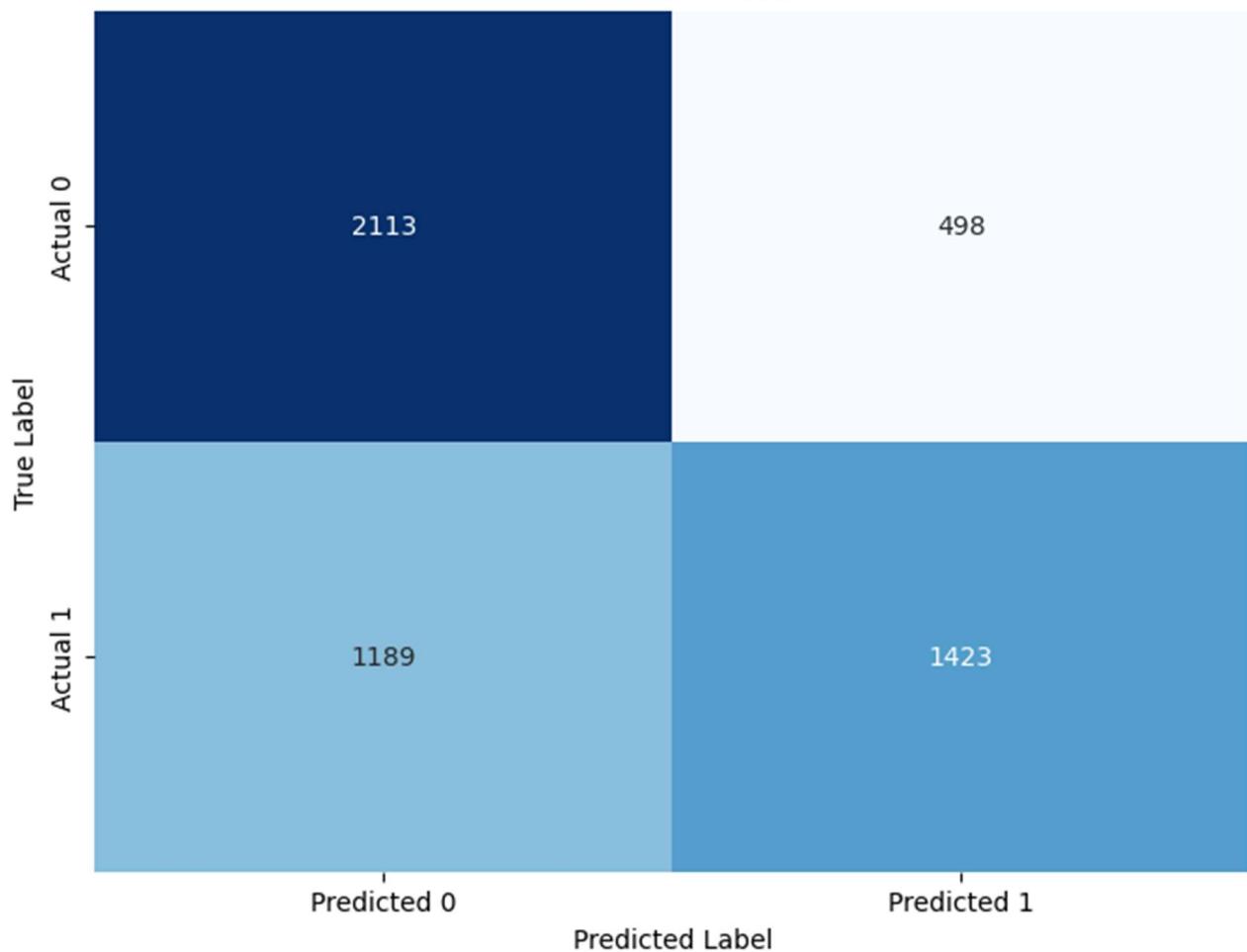
20% di sporcatura:

```
Model: GaussianNB
Accuracy test: 0.6770 | Accuracy Training: 0.6074
F1-score test: 0.6278 | F1-score Training: 0.5487
roc_AUC-score test: 0.7551 | roc_AUC-score Training: 0.6564
Differenza Accuracy: -0.0696
Differenza F1-score: -0.0791
Differenza AUC: -0.0987

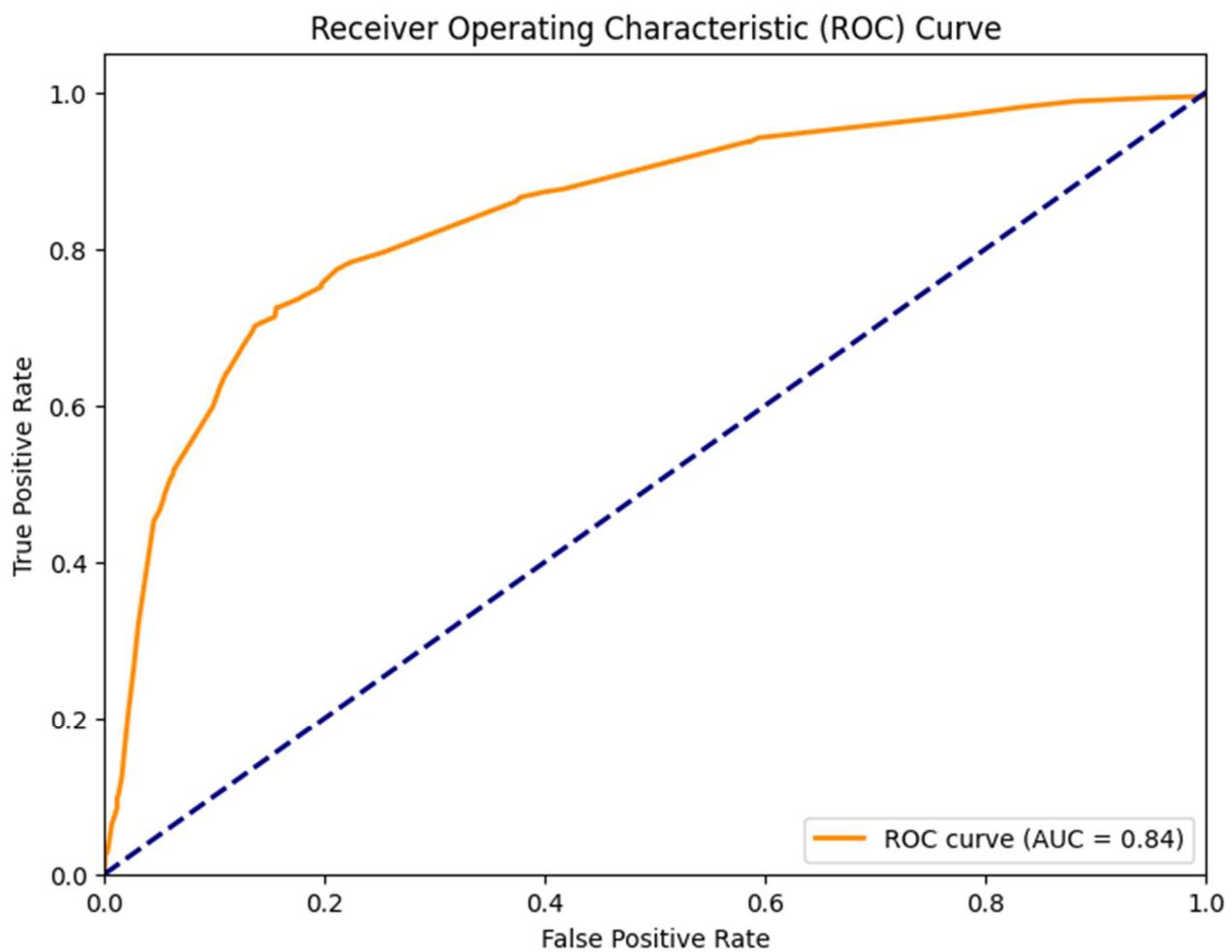
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'loan_int_rate': 0.0793
'credit_score': 0.0431
'loan_amnt': 0.0248
'person_income': 0.0195
'person_home_ownership_RENT': 0.0021
'person_age': 0.0020
'person_emp_exp': 0.0018
'loan_intent_VENTURE': 3.4463e-04
'cb_person_cred_hist_length': 3.0634e-04
'loan_intent_PERSONAL': 7.6584e-05
'loan_intent_EDUCATION': 7.6584e-05
'loan_intent_MEDICAL': 7.6584e-05
'person_education_Bachelor': 3.8292e-05
'person_education_Doctorate': 3.8292e-05
'person_home_ownership_OWN': -2.2204e-17
'person_education_High School': -3.4463e-04
AUC: 0.7551
```



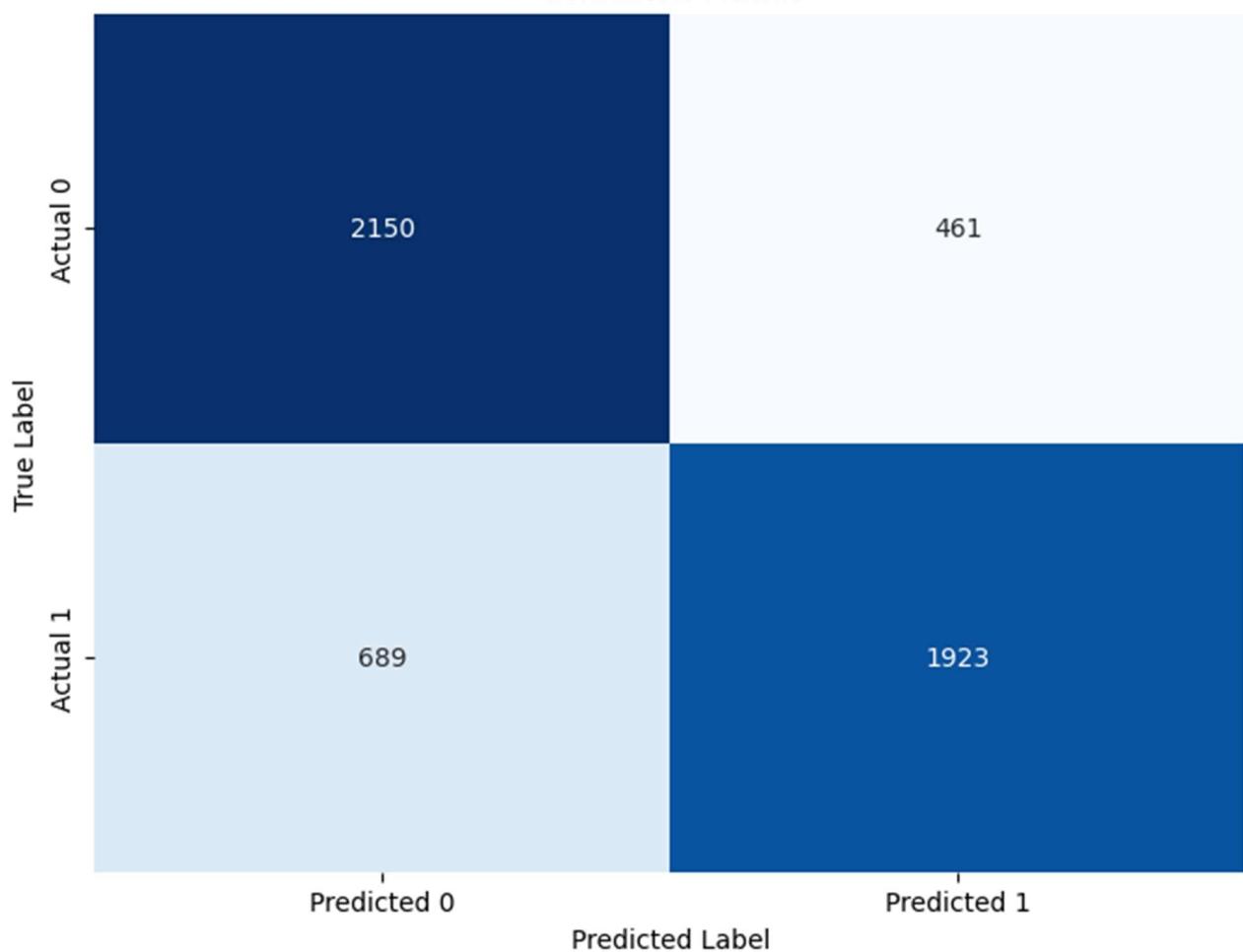
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.7798 | Accuracy Training: 0.6991
F1-score test: 0.7698 | F1-score Training: 0.6833
roc_AUC-score test: 0.8445 | roc_AUC-score Training: 0.7494
Differenza Accuracy: -0.0807
Differenza F1-score: -0.0865
Differenza AUC: -0.0951
Importanza delle Feature per Decision Tree:
'person_income': 0.4033
'loan_int_rate': 0.3282
'loan_amnt': 0.1734
'person_home_ownership_RENT': 0.0346
'credit_score': 0.0313
'loan_intent_HOMEIMPROVEMENT': 0.0099
'person_age': 0.0043
'person_home_ownership_OWN': 0.0039
'loan_intent_EDUCATION': 0.0029
'person_education_Document': 0.0028
'loan_intent_MEDICAL': 0.0027
'cb_person_cred_hist_length': 0.0026
AUC: 0.8445
```

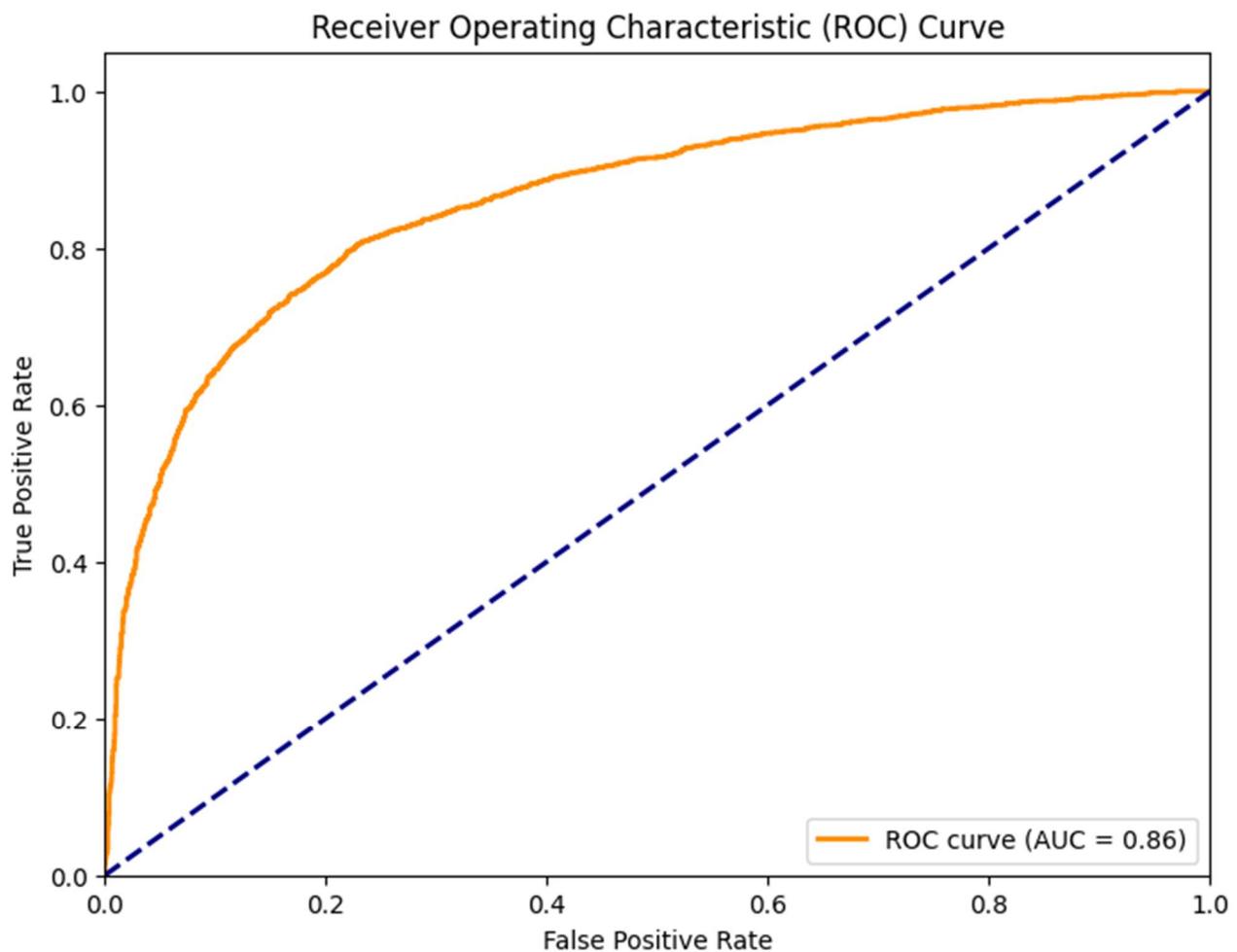


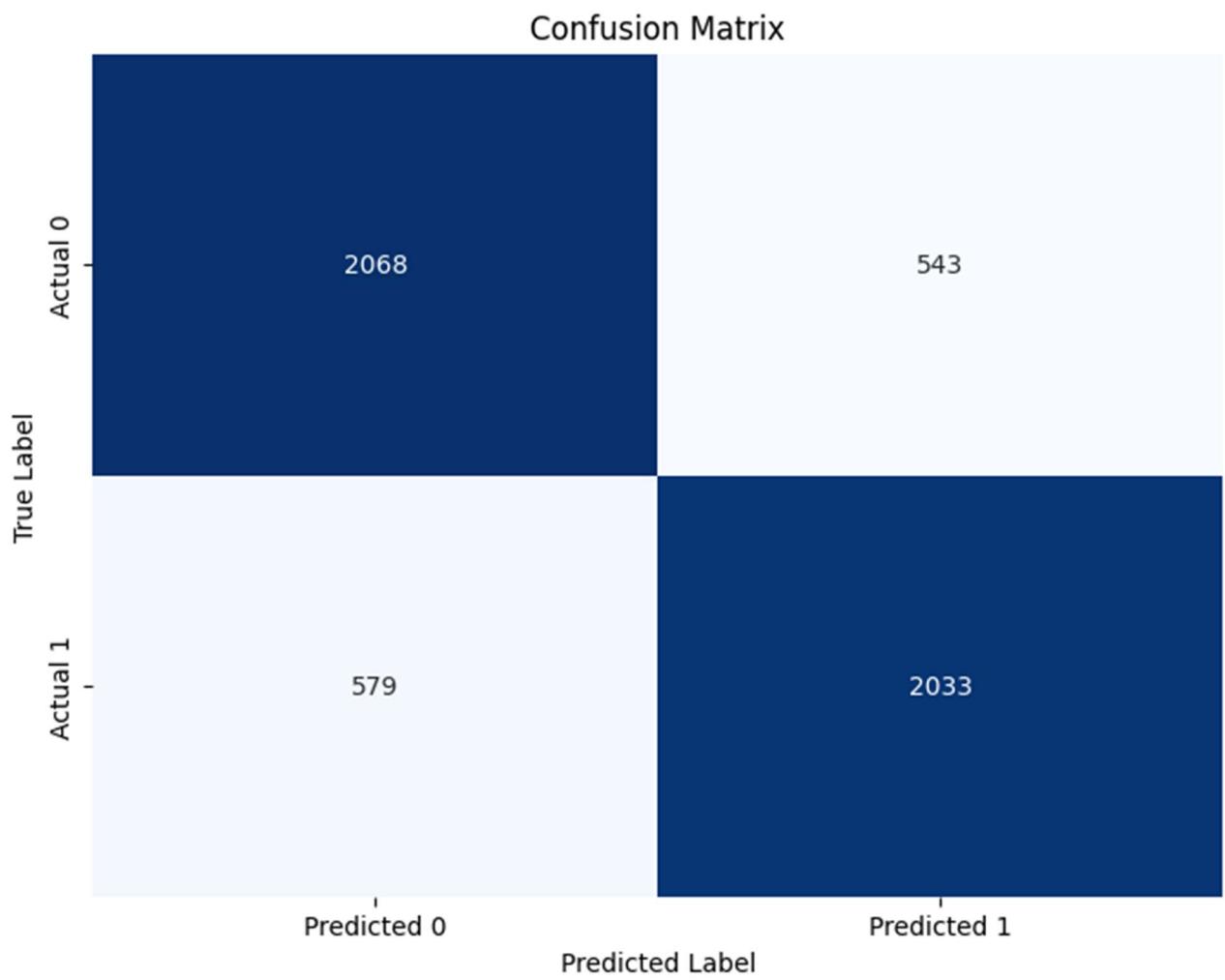
Confusion Matrix



```
Model: MLPClassifier
Accuracy test: 0.7852 | Accuracy Training: 0.7090
F1-score test: 0.7837 | F1-score Training: 0.7023
roc_AUC-score test: 0.8602 | roc_AUC-score Training: 0.7732
Differenza Accuracy: -0.0761
Differenza F1-score: -0.0814
Differenza AUC: -0.0870

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age          0.0
person_income        0.0
person_emp_exp       0.0
loan_amnt           0.0
loan_int_rate        0.0
cb_person_cred_hist_length 0.0
credit_score         0.0
person_gender_male   0.0
person_education_Bachelor 0.0
person_education_Doctorate 0.0
person_education_High School 0.0
person_education_Master 0.0
person_home_ownership_OTHER 0.0
person_home_ownership_OWN 0.0
person_home_ownership_RENT 0.0
loan_intent_EDUCATION 0.0
loan_intent_HOMEIMPROVEMENT 0.0
loan_intent_MEDICAL 0.0
loan_intent_PERSONAL 0.0
loan_intent_VENTURE 0.0
dtype: float64
AUC: 0.8602
```



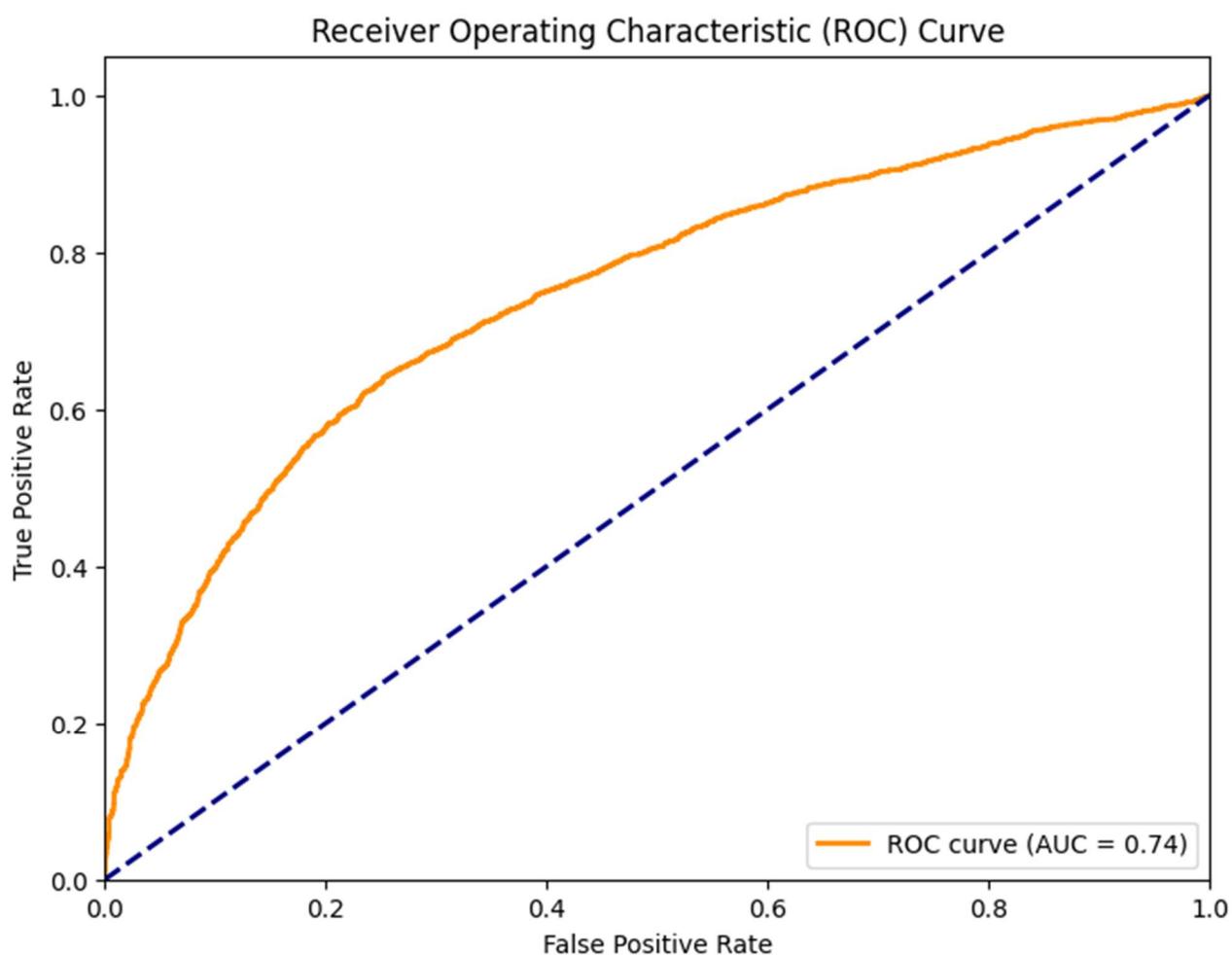


Al 20% è interessante notare come il Naive-Bayes riesce a classificare meglio i target 0 mentre ha un significativo peggioramento nella classificazione dei target 1, questa variazione potrebbe essere causata dal fatto che al 20% il rumore introdotto spinge il modello a classificare più spesso i valori come 0.

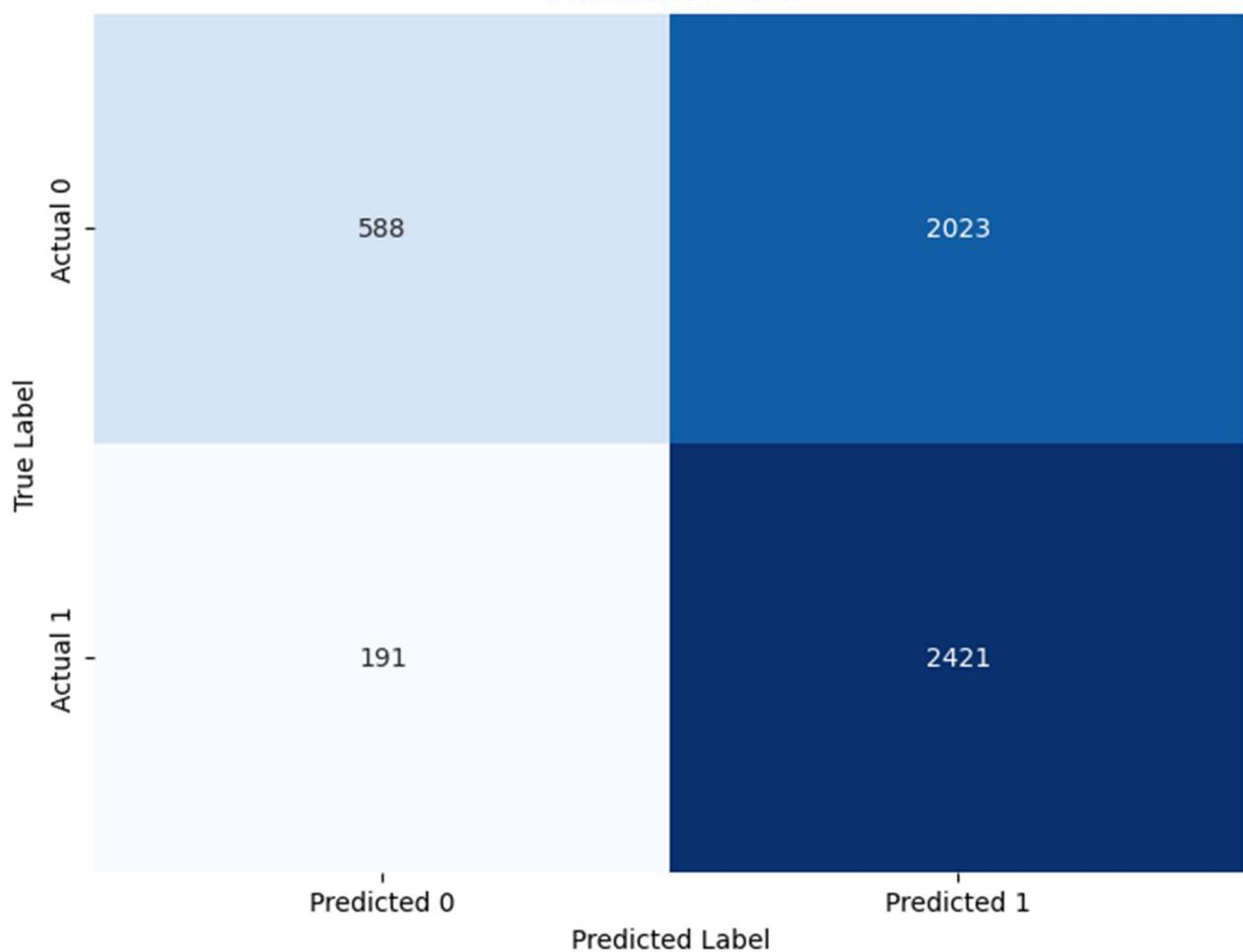
30% di sporcatura:

```
Model: GaussianNB
Accuracy test: 0.5761 | Accuracy Training: 0.5334
F1-score test: 0.6862 | F1-score Training: 0.6583
roc_AUC-score test: 0.7446 | roc_AUC-score Training: 0.6007
Differenza Accuracy: -0.0427
Differenza F1-score: -0.0279
Differenza AUC: -0.1439

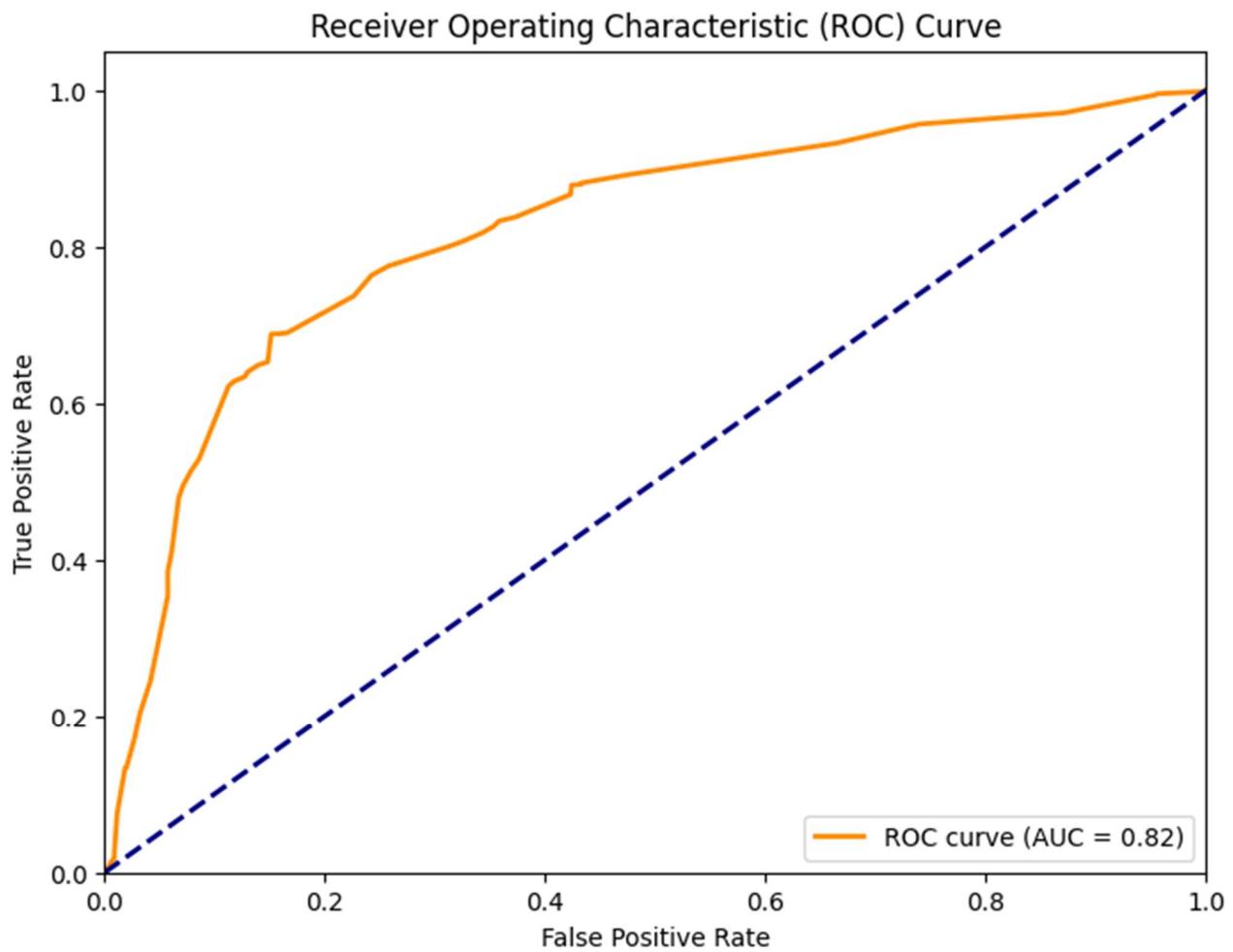
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'person_income': 0.0264
'loan_int_rate': 0.0228
'credit_score': 0.0101
'loan_amnt': 0.0015
'person_home_ownership_RENT': 0.0013
'loan_intent_MEDICAL': 2.2975e-04
'person_gender_male': 7.6584e-05
'loan_intent_PERSONAL': 2.2204e-17
'loan_intent_VENTURE': 2.2204e-17
'loan_intent_EDUCATION': -7.6584e-05
'person_education_High School': -1.1488e-04
'person_home_ownership_OWN': -1.1488e-04
'loan_intent_HOMEIMPROVEMENT': -1.5317e-04
'cb_person_cred_hist_length': -1.5317e-04
'person_emp_exp': -4.9780e-04
'person_age': -7.6584e-04
AUC: 0.7446
```



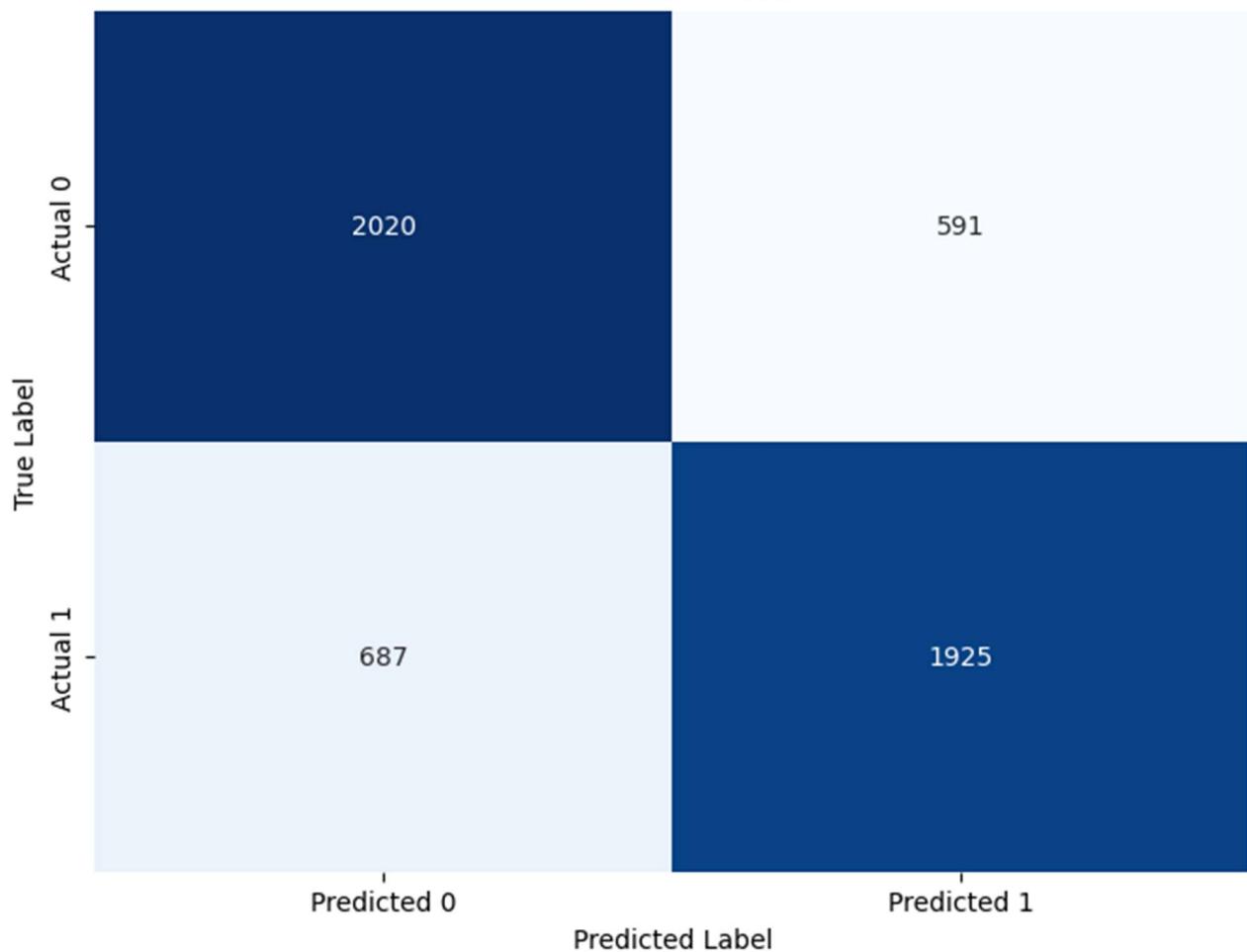
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.7553 | Accuracy Training: 0.6322
F1-score test: 0.7508 | F1-score Training: 0.6213
roc_AUC-score test: 0.8200 | roc_AUC-score Training: 0.6746
Differenza Accuracy: -0.1231
Differenza F1-score: -0.1295
Differenza AUC: -0.1454
Importanza delle Feature per Decision Tree:
'person_income': 0.3592
'loan_int_rate': 0.3351
'loan_amnt': 0.1503
'credit_score': 0.0891
'cb_person_cred_hist_length': 0.0168
'person_home_ownership_RENT': 0.0141
'loan_intent_HOMEIMPROVEMENT': 0.0076
'person_home_ownership_OWN': 0.0071
'loan_intent_PERSONAL': 0.0069
'loan_intent_MEDICAL': 0.0067
'person_age': 0.0042
'person_education_Bachelor': 0.0028
AUC: 0.8200
```

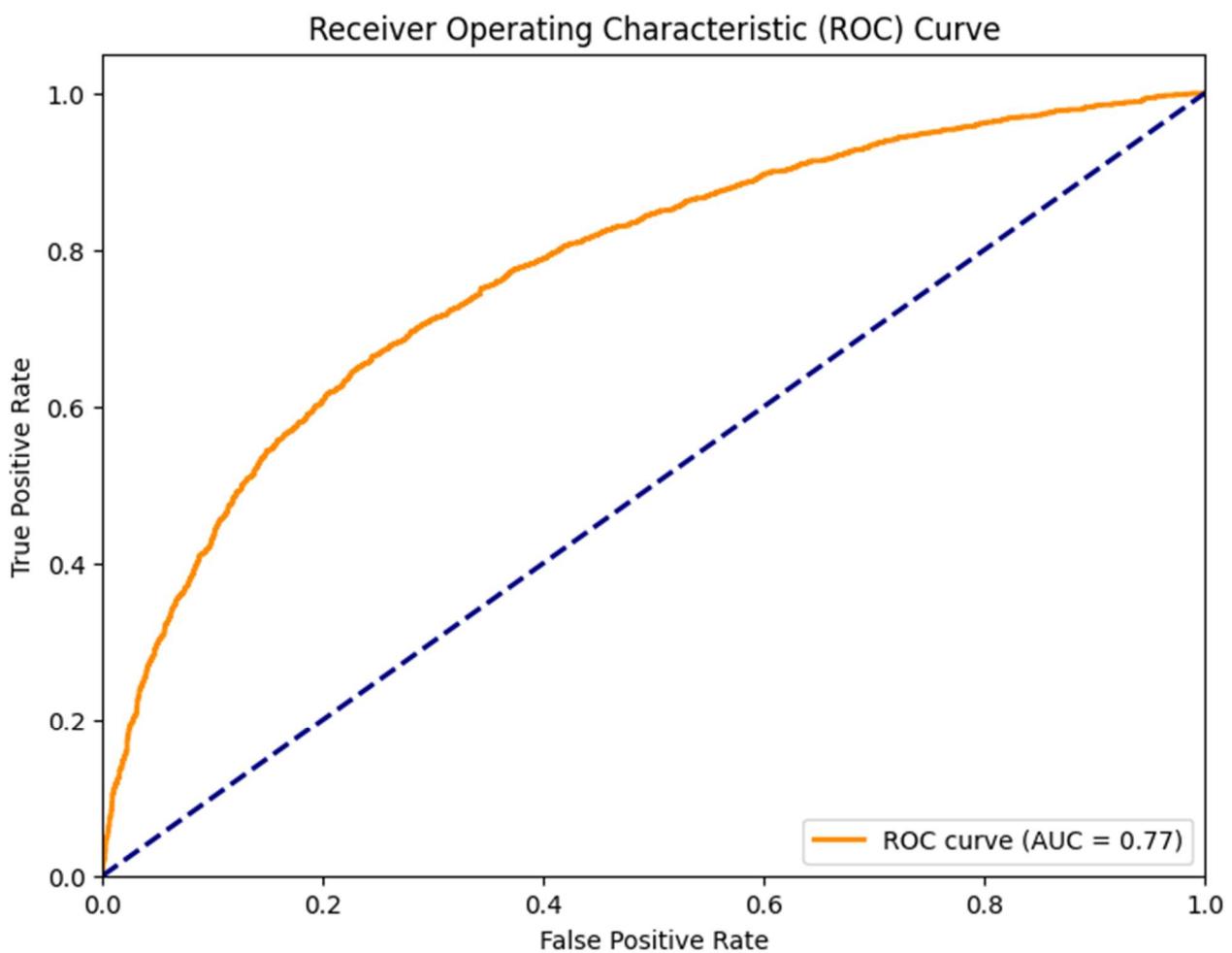


Confusion Matrix



```
Model: MLPClassifier
Accuracy test: 0.7082 | Accuracy Training: 0.6500
F1-score test: 0.6916 | F1-score Training: 0.6339
roc_AUC-score test: 0.7738 | roc_AUC-score Training: 0.7086
Differenza Accuracy: -0.0583
Differenza F1-score: -0.0577
Differenza AUC: -0.0652

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age          0.0
person_income        0.0
person_emp_exp       0.0
loan_amnt           0.0
loan_int_rate        0.0
cb_person_cred_hist_length 0.0
credit_score         0.0
person_gender_male   0.0
person_education_Bachelor 0.0
person_education_Doctorate 0.0
person_education_High School 0.0
person_education_Master 0.0
person_home_ownership_OTHER 0.0
person_home_ownership_OWN 0.0
person_home_ownership_RENT 0.0
loan_intent_EDUCATION 0.0
loan_intent_HOMEIMPROVEMENT 0.0
loan_intent_MEDICAL 0.0
loan_intent_PERSONAL 0.0
loan_intent_VENTURE 0.0
dtype: float64
AUC: 0.7738
```



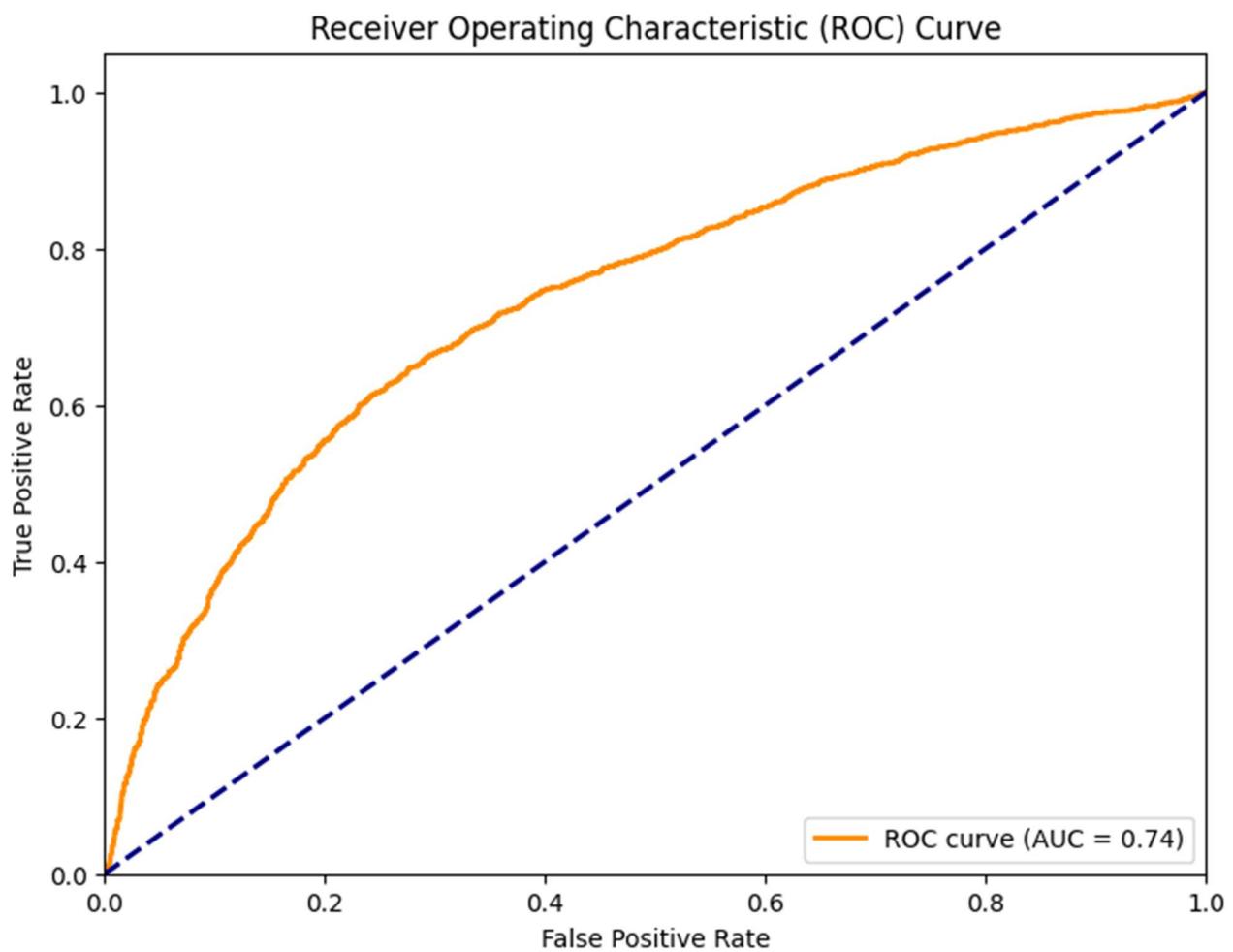
Confusion Matrix

		Predicted Label
True Label	Actual 0	Actual 1
	Predicted 0	Predicted 1
Actual 0	1990	621
Actual 1	903	1709

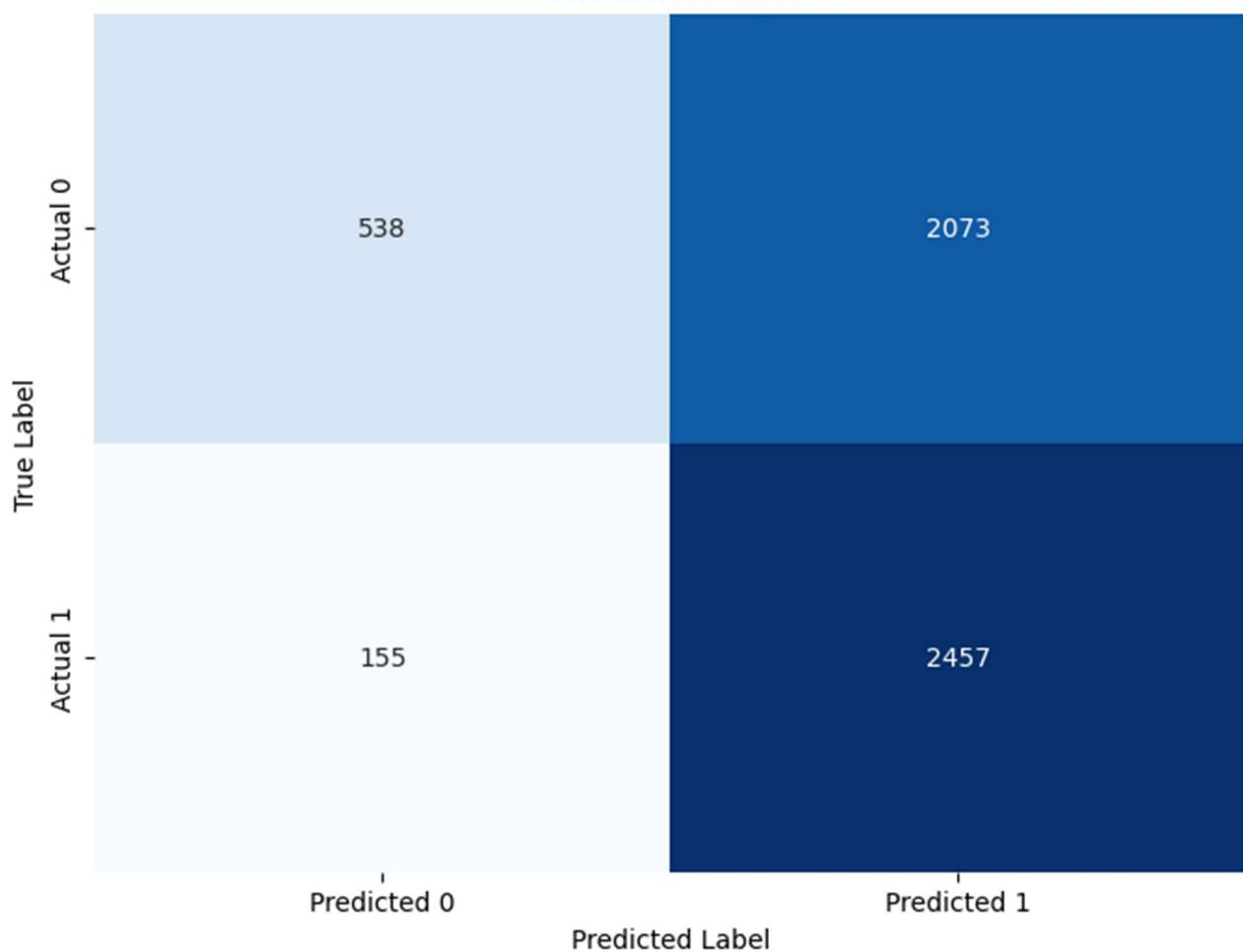
40% di sporcatura:

```
Model: GaussianNB
Accuracy test: 0.5734 | Accuracy Training: 0.5169
F1-score test: 0.6880 | F1-score Training: 0.6484
roc_AUC-score test: 0.7352 | roc_AUC-score Training: 0.5514
Differenza Accuracy: -0.0566
Differenza F1-score: -0.0396
Differenza AUC: -0.1838

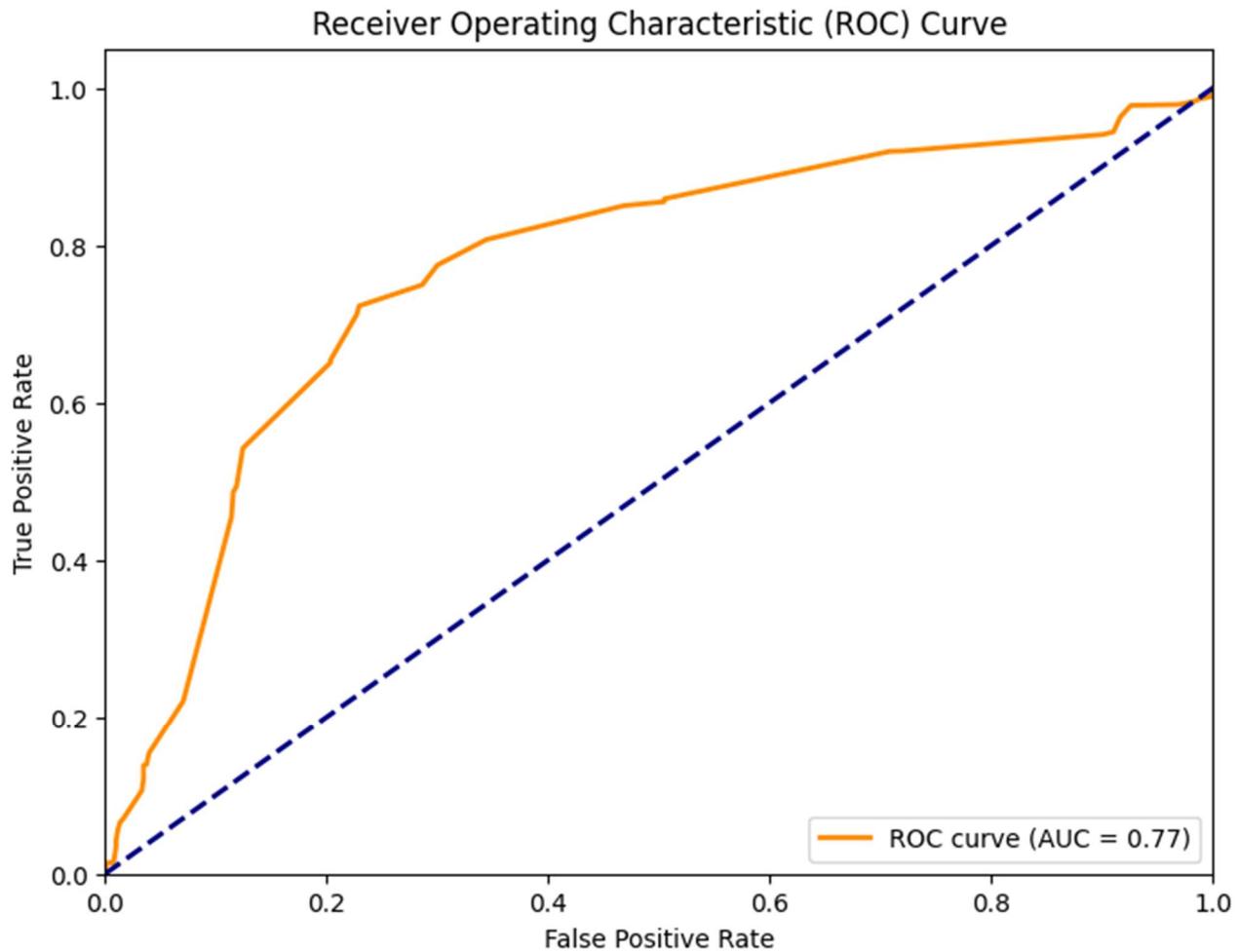
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'loan_int_rate': 0.0305
'person_income': 0.0224
'credit_score': 0.0102
'person_home_ownership_RENT': 0.0013
'cb_person_cred_hist_length': 0.0012
'person_age': 0.0010
'person_emp_exp': 6.8926e-04
'loan_intent_MEDICAL': 1.1488e-04
'person_education_Master': 3.8292e-05
'person_education_Bachelor': 3.8292e-05
'loan_intent_VENTURE': -1.5317e-04
'loan_intent_HOMEIMPROVEMENT': -2.6805e-04
'person_home_ownership_OWN': -3.0634e-04
'loan_intent_PERSONAL': -3.8292e-04
'person_gender_male': -4.5951e-04
'loan_amnt': -0.0025
AUC: 0.7352
```



Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.7317 | Accuracy Training: 0.5837
F1-score test: 0.7365 | F1-score Training: 0.5863
roc_AUC-score test: 0.7714 | roc_AUC-score Training: 0.6132
Differenza Accuracy: -0.1480
Differenza F1-score: -0.1502
Differenza AUC: -0.1582
Importanza delle Feature per Decision Tree:
'person_income': 0.3425
'loan_amnt': 0.1587
'loan_int_rate': 0.1564
'credit_score': 0.0995
'person_emp_exp': 0.0547
'person_education_High School': 0.0401
'cb_person_cred_hist_length': 0.0388
'person_age': 0.0358
'loan_intent_HOMEIMPROVEMENT': 0.0177
'person_home_ownership_OWN': 0.0156
'loan_intent_EDUCATION': 0.0136
'person_gender_male': 0.0118
'loan_intent_VENTURE': 0.0102
'loan_intent_PERSONAL': 0.0049
AUC: 0.7714
```

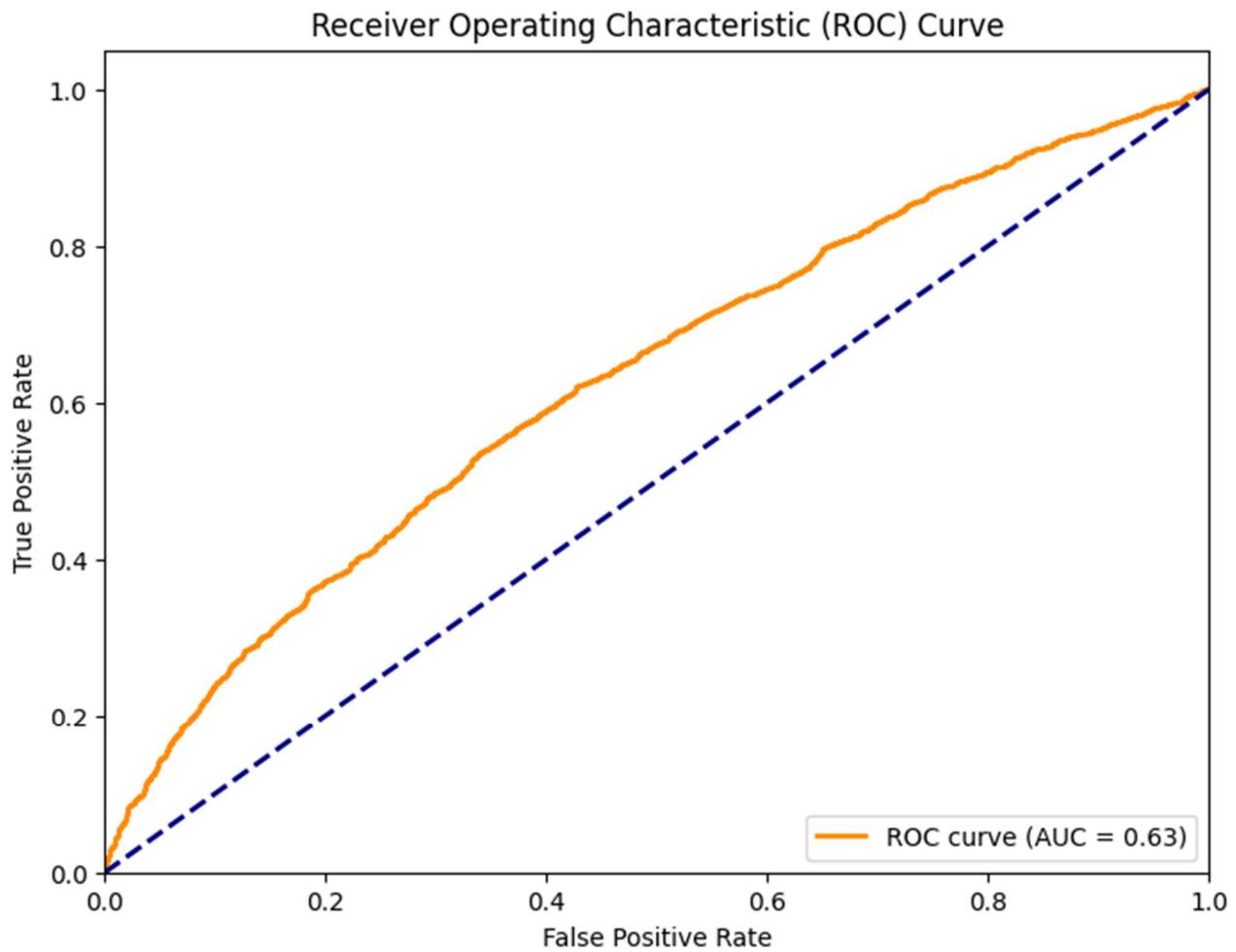


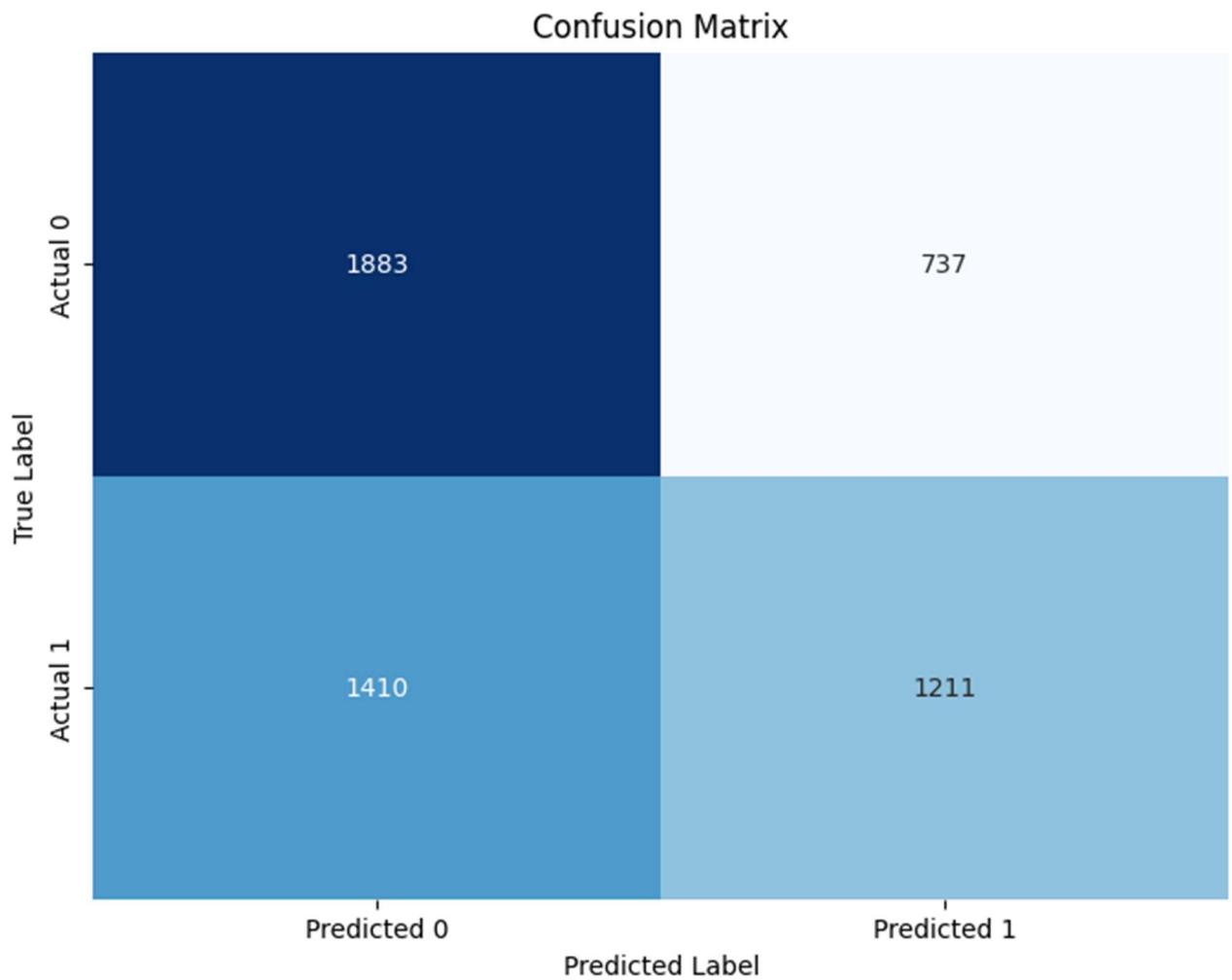
Confusion Matrix

		Predicted Label
True Label	Actual 0	1870
	Predicted 1	750
Actual 1	656	1965

```
Model: MLPClassifier
Accuracy test: 0.5903 | Accuracy Training: 0.6166
F1-score test: 0.5301 | F1-score Training: 0.5721
roc_AUC-score test: 0.6289 | roc_AUC-score Training: 0.6673
Differenza Accuracy: 0.0263
Differenza F1-score: 0.0420
Differenza AUC: 0.0384

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age 0.0
person_income 0.0
person_emp_exp 0.0
loan_amnt 0.0
loan_int_rate 0.0
cb_person_cred_hist_length 0.0
credit_score 0.0
person_gender_male 0.0
person_education_Bachelor 0.0
person_education_Doctorate 0.0
person_education_High School 0.0
person_education_Master 0.0
person_home_ownership_OTHER 0.0
person_home_ownership_OWN 0.0
person_home_ownership_RENT 0.0
loan_intent_EDUCATION 0.0
loan_intent_HOMEIMPROVEMENT 0.0
loan_intent_MEDICAL 0.0
loan_intent_PERSONAL 0.0
loan_intent_VENTURE 0.0
dtype: float64
AUC: 0.6289
```





È interessante notare come al 30% il naive bayes ritorna a valori coerenti con quelli del dataset pulito mentre sia al 20 che al 30% l'albero decisionale ed il MLP rimangono solidi e con valori relativamente costanti.

Al 40% sia il MLPClassifier che l'albero decisionale crollano mostrando seri casi di underfitting del modello.

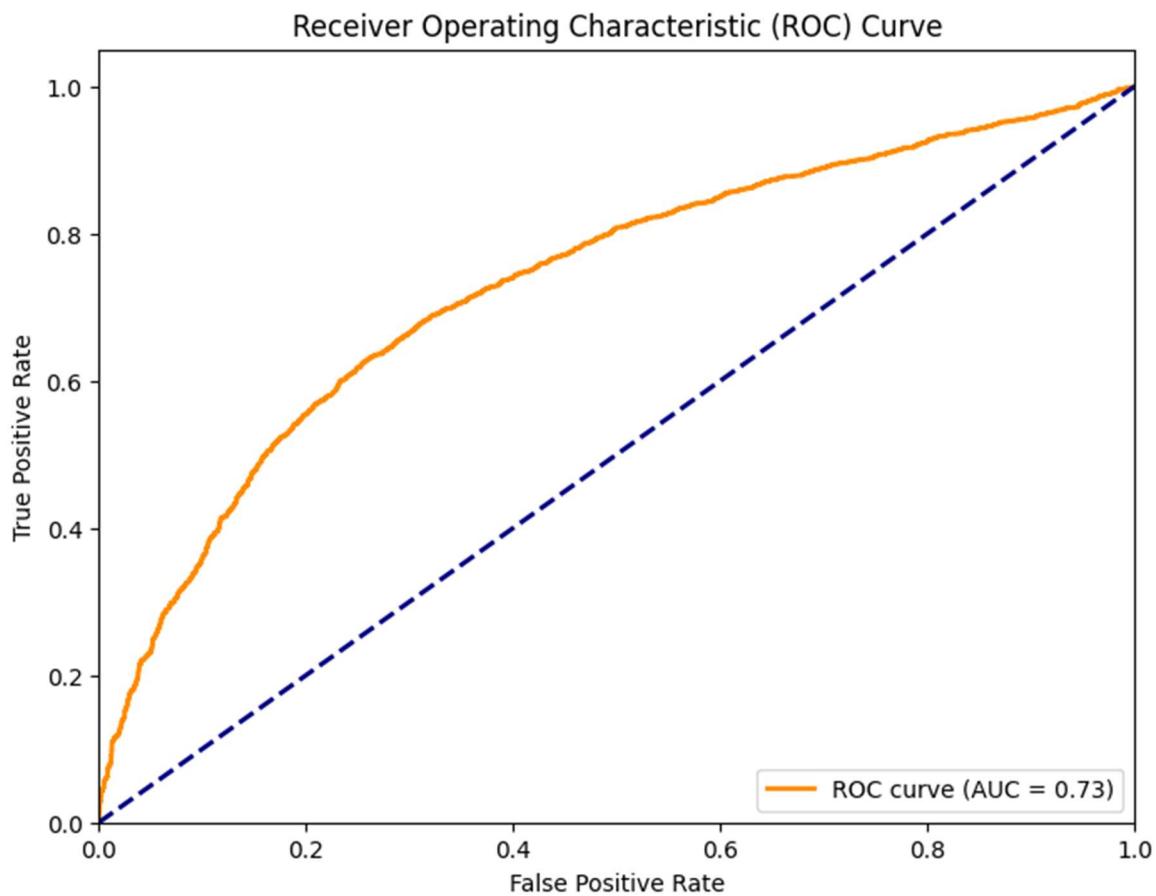
Notare che pure la confusion matrix del MLPClassifier risulta particolarmente errata.

Errori di input manuali 20%,80%

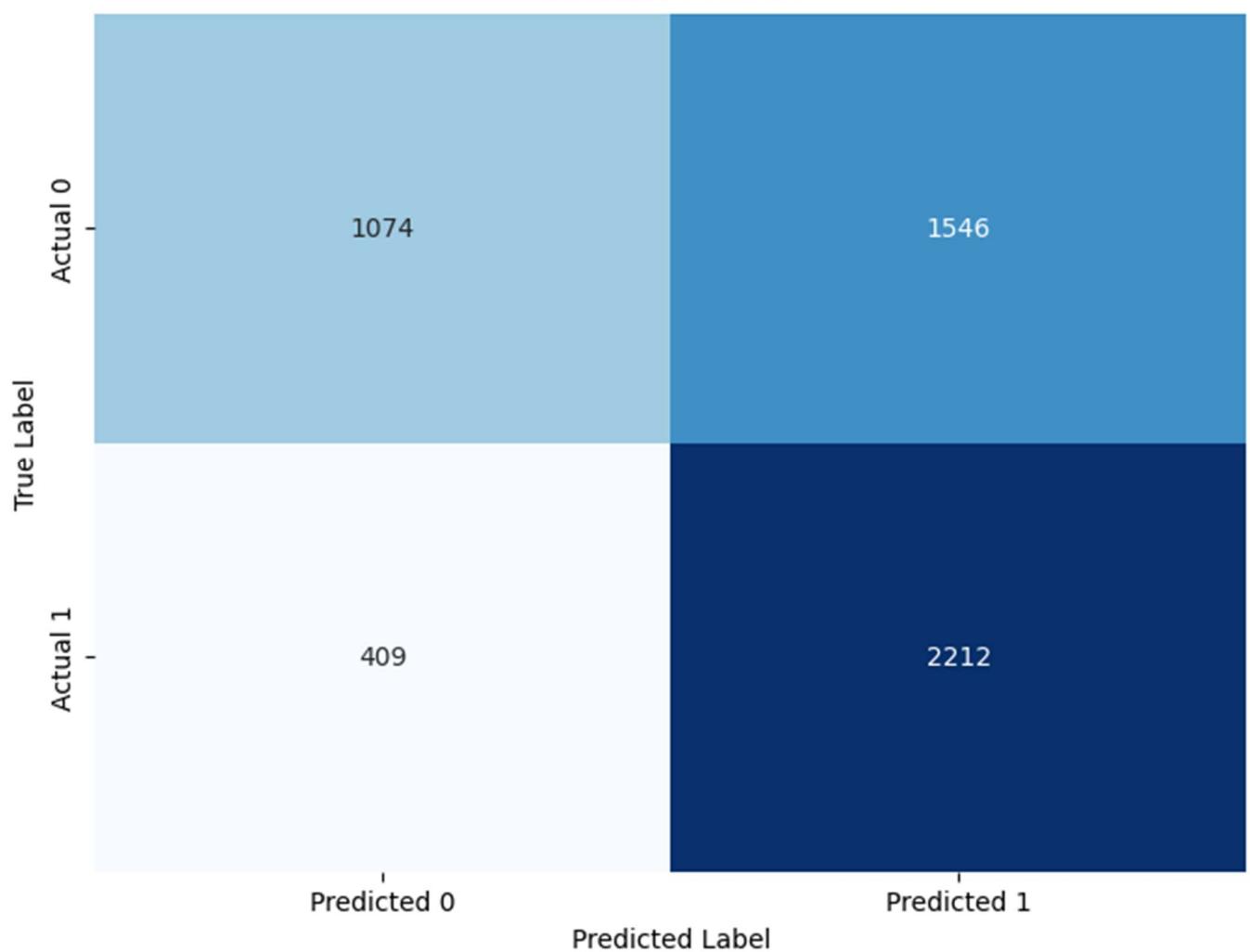
20%

```
Model: GaussianNB
Accuracy test: 0.6270 | Accuracy Training: 0.6351
F1-score test: 0.6935 | F1-score Training: 0.7023
roc_AUC-score test: 0.7312 | roc_AUC-score Training: 0.7429
Differenza Accuracy: 0.0082
Differenza F1-score: 0.0087
Differenza AUC: 0.0116

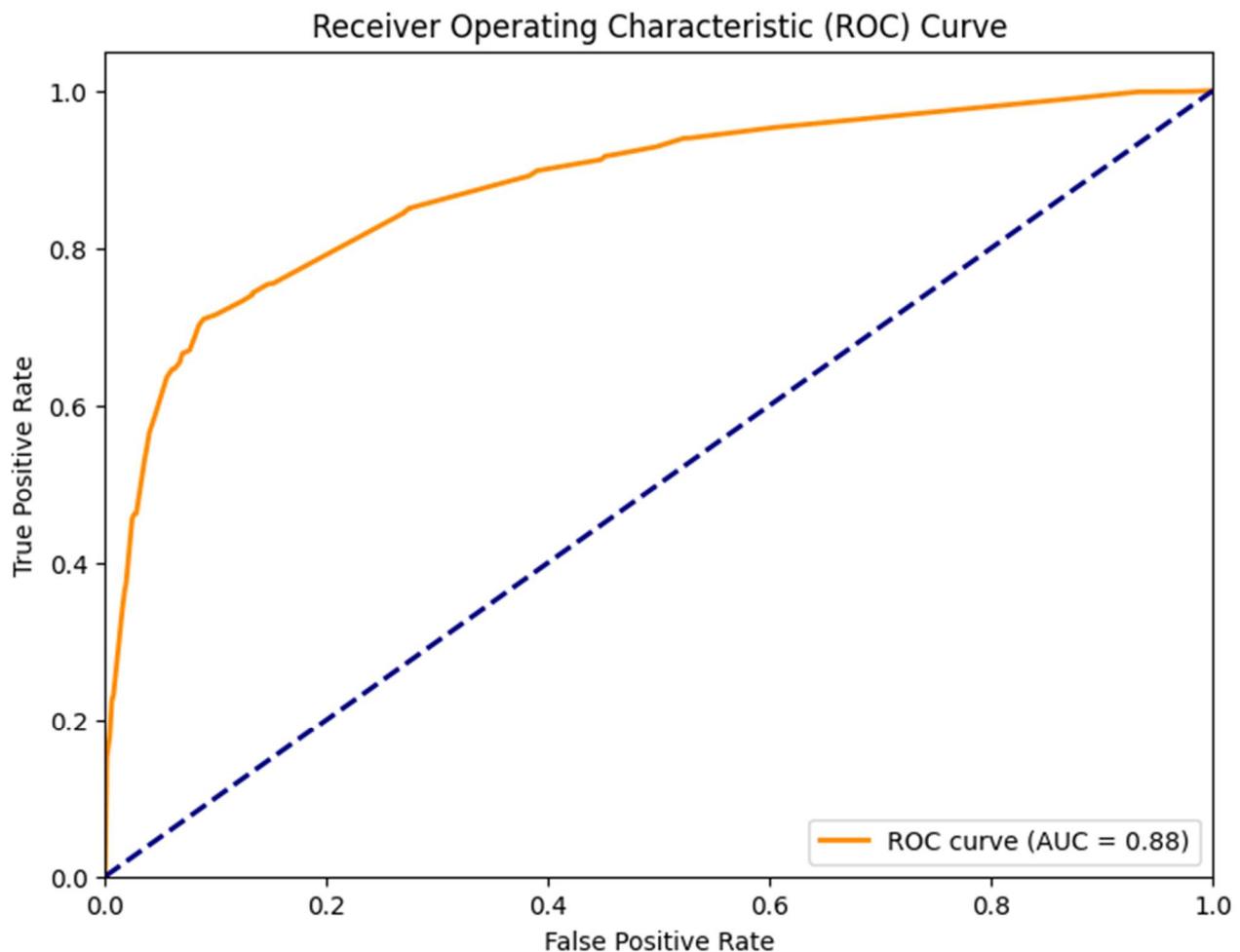
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'loan_int_rate': 0.0532
'person_income': 0.0392
'credit_score': 0.0133
'loan_amnt': 0.0098
'person_home_ownership_RENT': 0.0012
'loan_intent_VENTURE': 3.4345e-04
'person_education_High School': 2.6712e-04
'loan_intent_EDUCATION': 1.9080e-04
'person_education_Bachelor': 1.5264e-04
'person_home_ownership_OWN': 1.1448e-04
'loan_intent_MEDICAL': 3.8161e-05
'person_education_Doctorate': -3.8161e-05
'person_education_Master': -1.1448e-04
'loan_intent_HOMEIMPROVEMENT': -1.5264e-04
'loan_intent_PERSONAL': -2.2896e-04
'cb_person_cred_hist_length': -0.0064
'person_age': -0.0084
'person_emp_exp': -0.0126
AUC: 0.7312
```



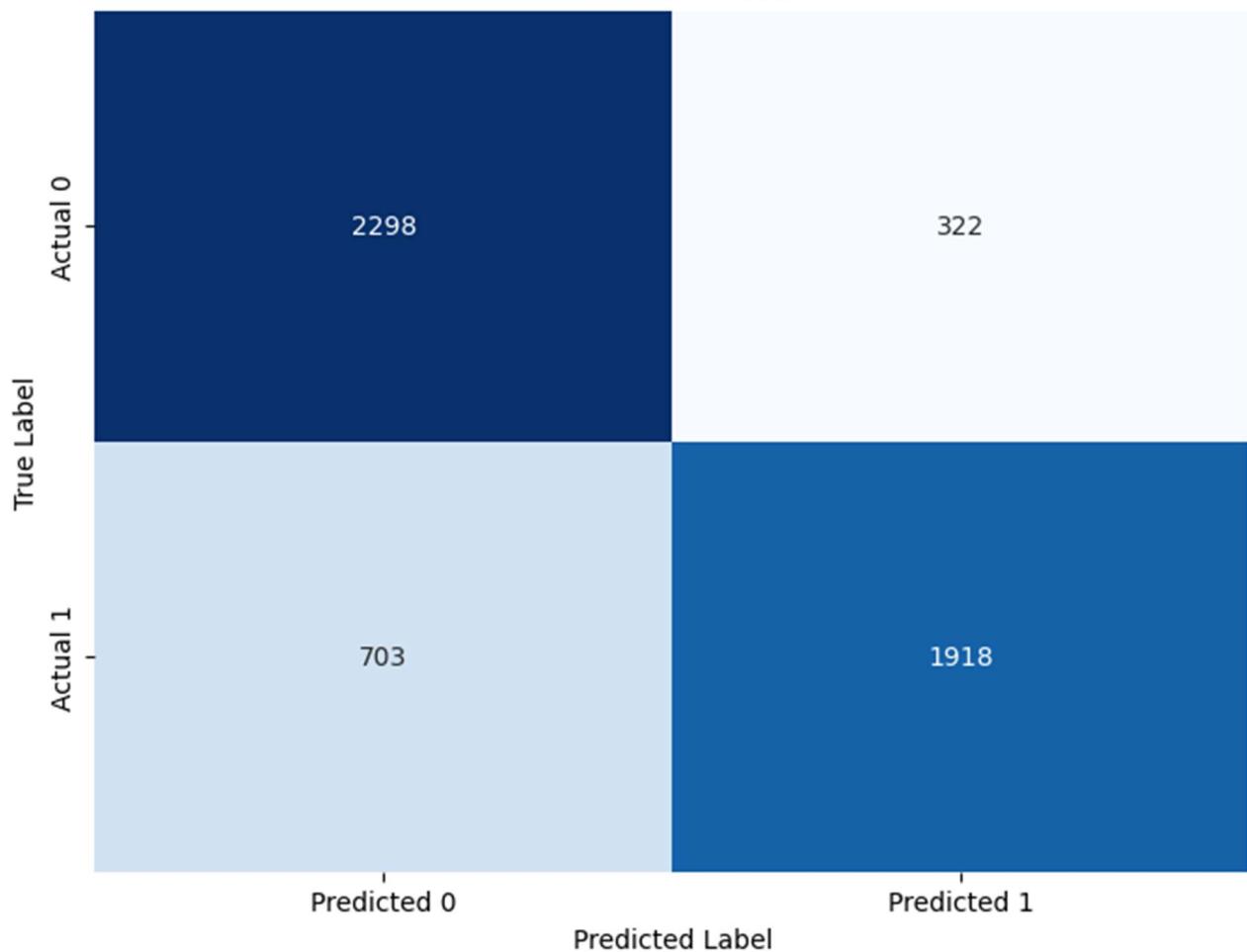
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.8044 | Accuracy Training: 0.8145
F1-score test: 0.7891 | F1-score Training: 0.7999
roc_AUC-score test: 0.8781 | roc_AUC-score Training: 0.8914
Differenza Accuracy: 0.0101
Differenza F1-score: 0.0107
Differenza AUC: 0.0133
Importanza delle Feature per Decision Tree:
'person_income': 0.3486
'loan_int_rate': 0.2915
'loan_amnt': 0.1348
'person_home_ownership_RENT': 0.1307
'credit_score': 0.0264
'person_home_ownership_OWN': 0.0159
'loan_intent_HOMEIMPROVEMENT': 0.0145
'loan_intent_MEDICAL': 0.0138
'loan_intent_VENTURE': 0.0068
'loan_intent_PERSONAL': 0.0063
'loan_intent_EDUCATION': 0.0059
'person_age': 0.0027
'cb_person_cred_hist_length': 0.0017
'person_education_Doctorate': 2.9884e-04
AUC: 0.8781
```

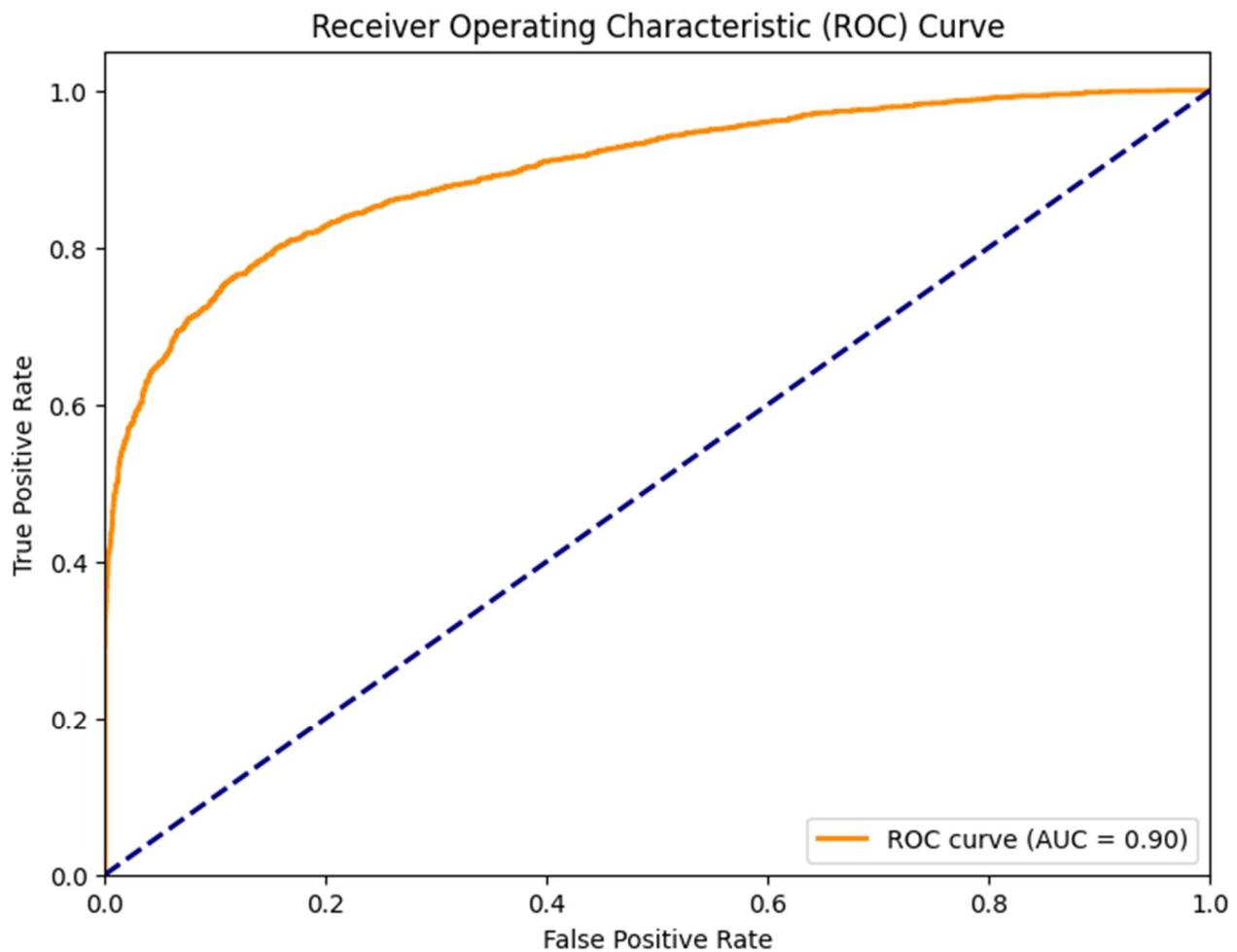


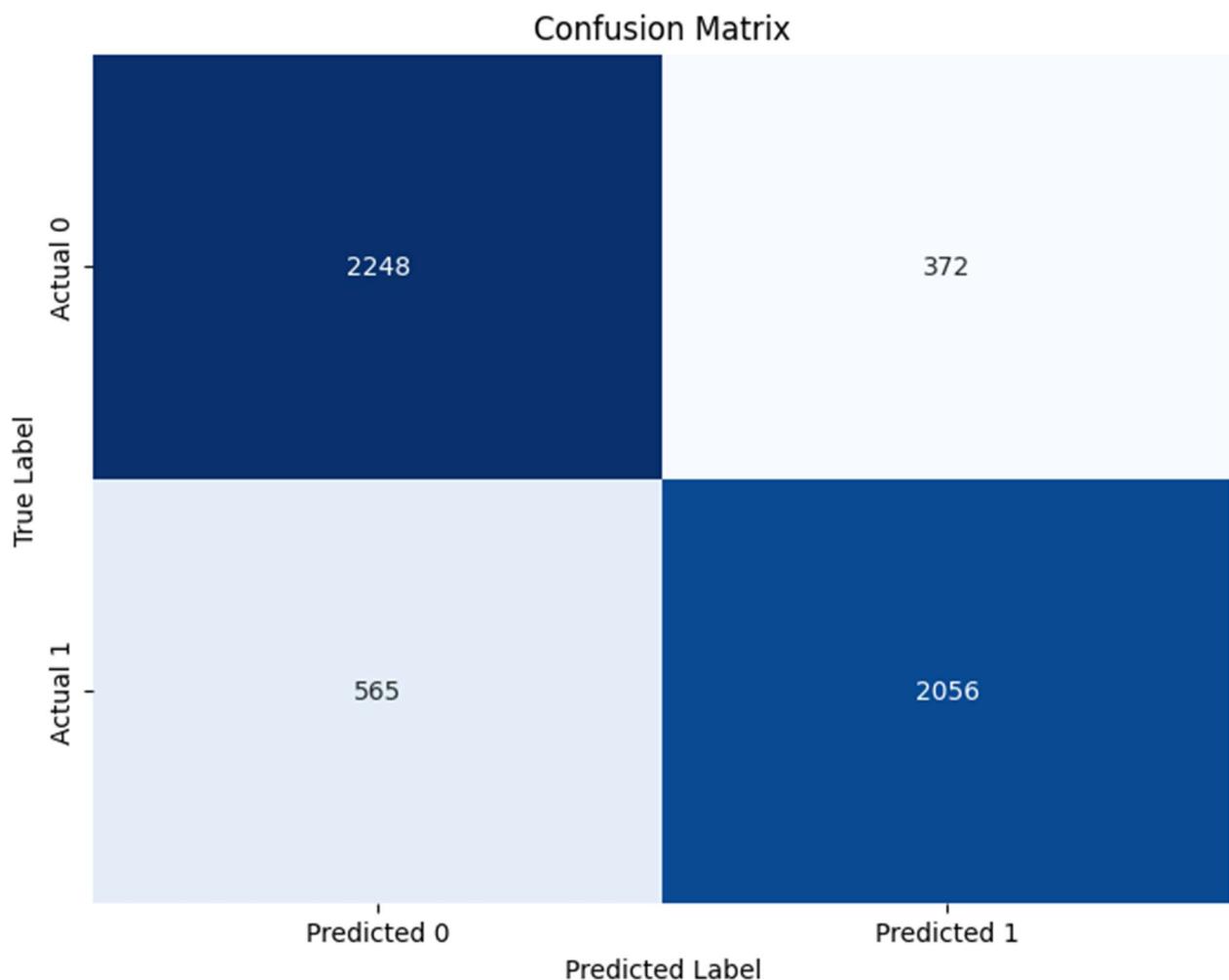
Confusion Matrix



```
Model: MLPClassifier
Accuracy test: 0.8212 | Accuracy Training: 0.8376
F1-score test: 0.8144 | F1-score Training: 0.8320
roc_AUC-score test: 0.8981 | roc_AUC-score Training: 0.9252
Differenza Accuracy: 0.0164
Differenza F1-score: 0.0176
Differenza AUC: 0.0271

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age          0.000000
person_emp_exp       0.000000
cb_person_cred_hist_length 0.000000
loan_int_rate         0.000000
person_home_ownership_OTHER 0.000000
person_home_ownership_OWN   0.000000
credit_score          0.000000
person_gender_male    0.000000
person_education_Bachelor 0.000000
person_education_Doctorate 0.000000
person_education_High School 0.000000
person_education_Master 0.000000
loan_intent_HOMEIMPROVEMENT 0.000000
loan_intent_MEDICAL    0.000000
person_home_ownership_RENT 0.000000
loan_intent_EDUCATION   0.000000
loan_intent_PERSONAL    0.000000
loan_intent_VENTURE     0.000000
person_income          -0.000038
loan_amnt              -0.000153
dtype: float64
AUC: 0.8981
```





A questo livello di sporcatura l'albero decisionale e MLP risultano relativamente costanti nella capacità predittiva.

Naive-Bayes, invece, sembra avere un miglioramento nell' identificazione dei target 0.

Questa variazione è probabilmente causata dagli "errori di battitura" delle variabili categoriche. Infatti, con l'introduzione della sporcatura possono esser state create nuove variazioni nei dati. Anche se queste variazioni sono "rumore", il modello Naive Bayes potrebbe averle interpretate come nuove "feature" indipendenti.

Se queste nuove "feature" introdotte per caso si sono correlate con la classe target=0 nel training set, il modello potrebbe averle utilizzate per migliorare la sua capacità di distinguere questa classe.

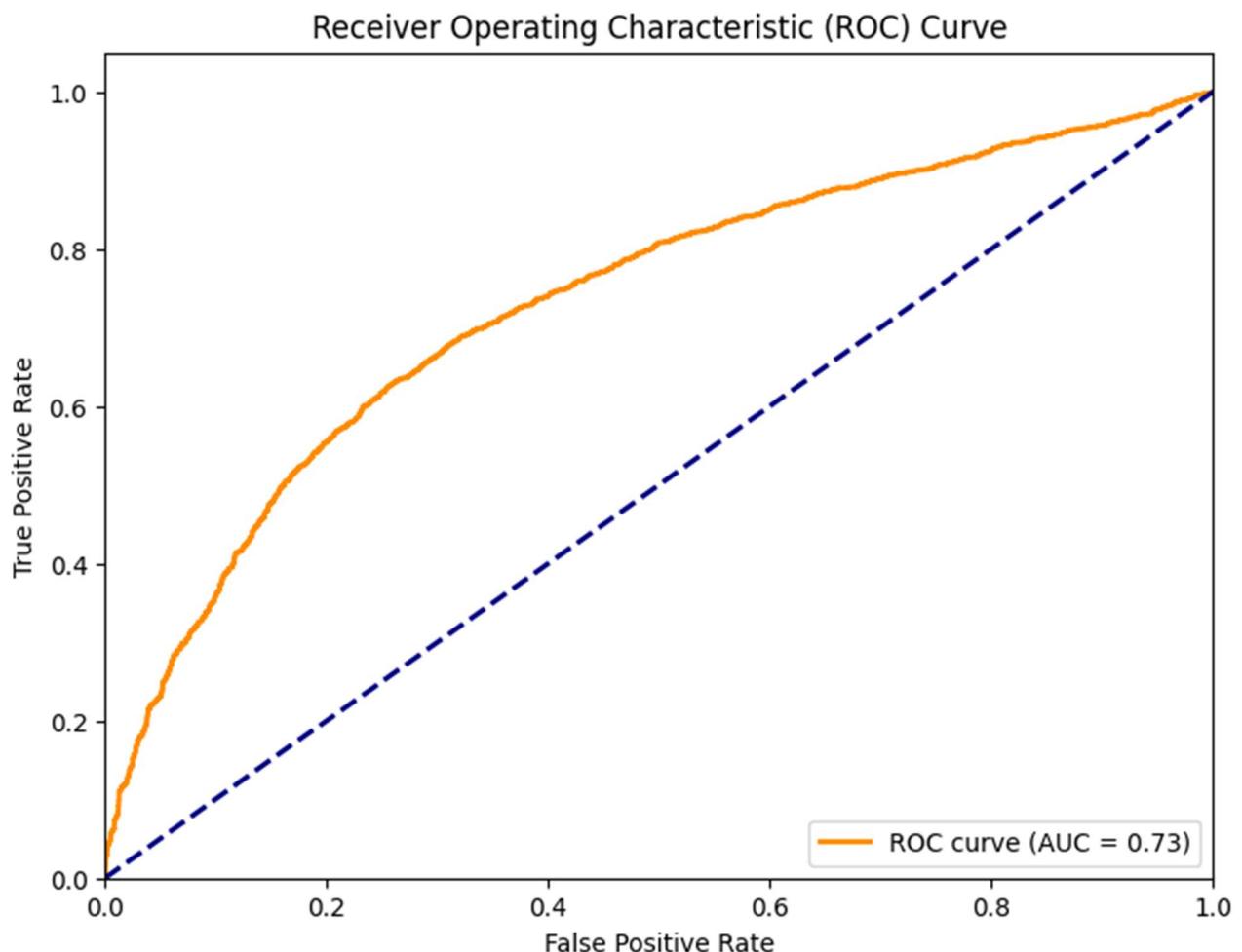
Ulteriore evidenza di ciò sta nel fatto che anche a livelli maggiori di sporcatura tutti e tre i modelli rimangono relativamente costanti nella loro capacità di classificazione.

In seguito, riporto il livello di sporcatura all' 80%.

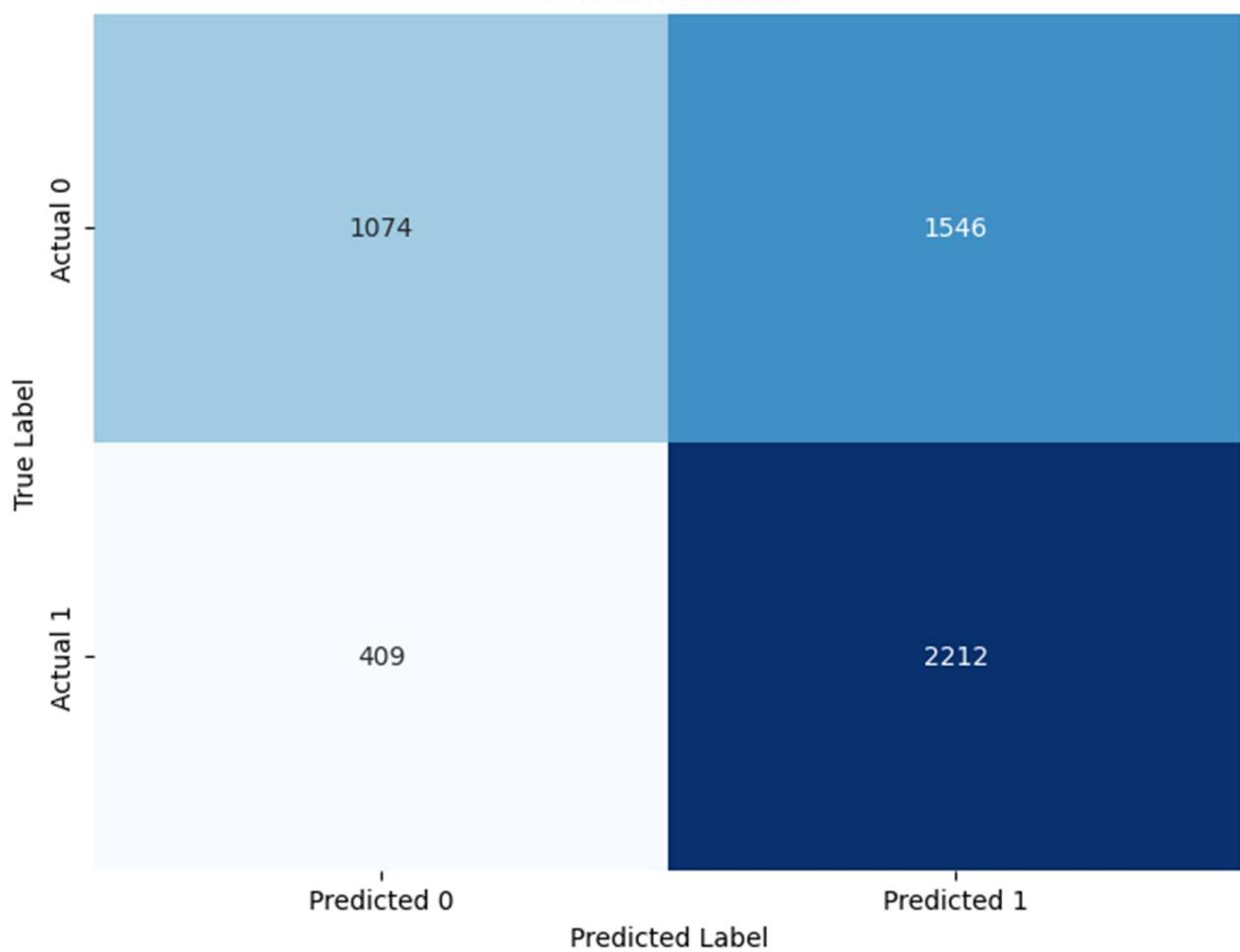
livello di sporcatura all' 80%:

```
Model: GaussianNB
Accuracy test: 0.6270 | Accuracy Training: 0.6351
F1-score test: 0.6935 | F1-score Training: 0.7023
roc_AUC-score test: 0.7312 | roc_AUC-score Training: 0.7429
Differenza Accuracy: 0.0082
Differenza F1-score: 0.0087
Differenza AUC: 0.0116

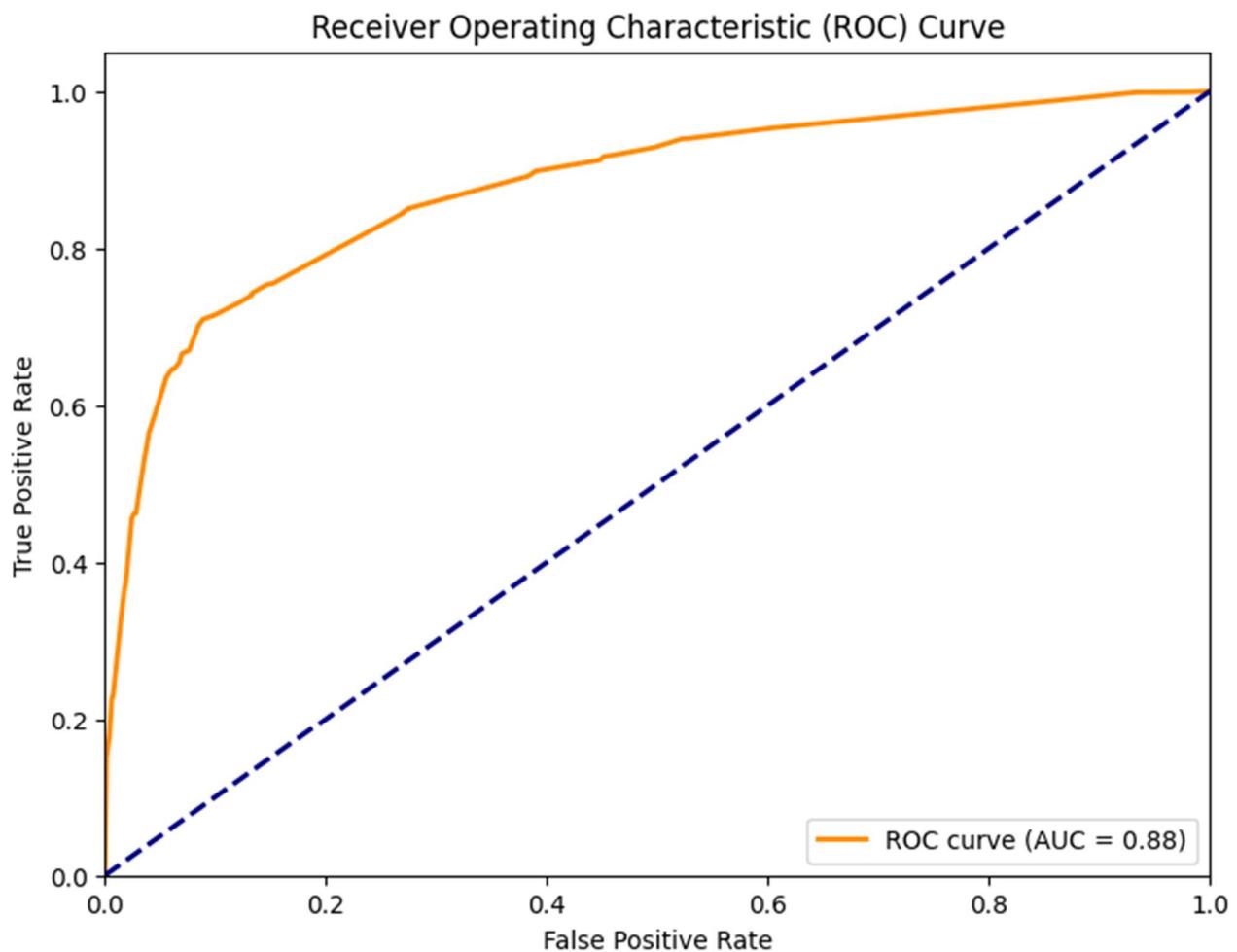
Importanza delle Feature (stimata con Permutation Importance) per Gaussian Naive Bayes:
'loan_int_rate': 0.0532
'person_income': 0.0392
'credit_score': 0.0133
'loan_amnt': 0.0098
'person_home_ownership_RENT': 0.0012
'loan_intent_VENTURE': 3.4345e-04
'person_education_High School': 2.6712e-04
'loan_intent_EDUCATION': 1.9080e-04
'person_education_Bachelor': 1.5264e-04
'person_home_ownership_OWN': 1.1448e-04
'loan_intent_MEDICAL': 3.8161e-05
'person_education_Document': -3.8161e-05
'person_education_Master': -1.1448e-04
'loan_intent_HOMEIMPROVEMENT': -1.5264e-04
'loan_intent_PERSONAL': -2.2896e-04
'cb_person_cred_hist_length': -0.0064
'person_age': -0.0084
'person_emp_exp': -0.0126
AUC: 0.7312
```



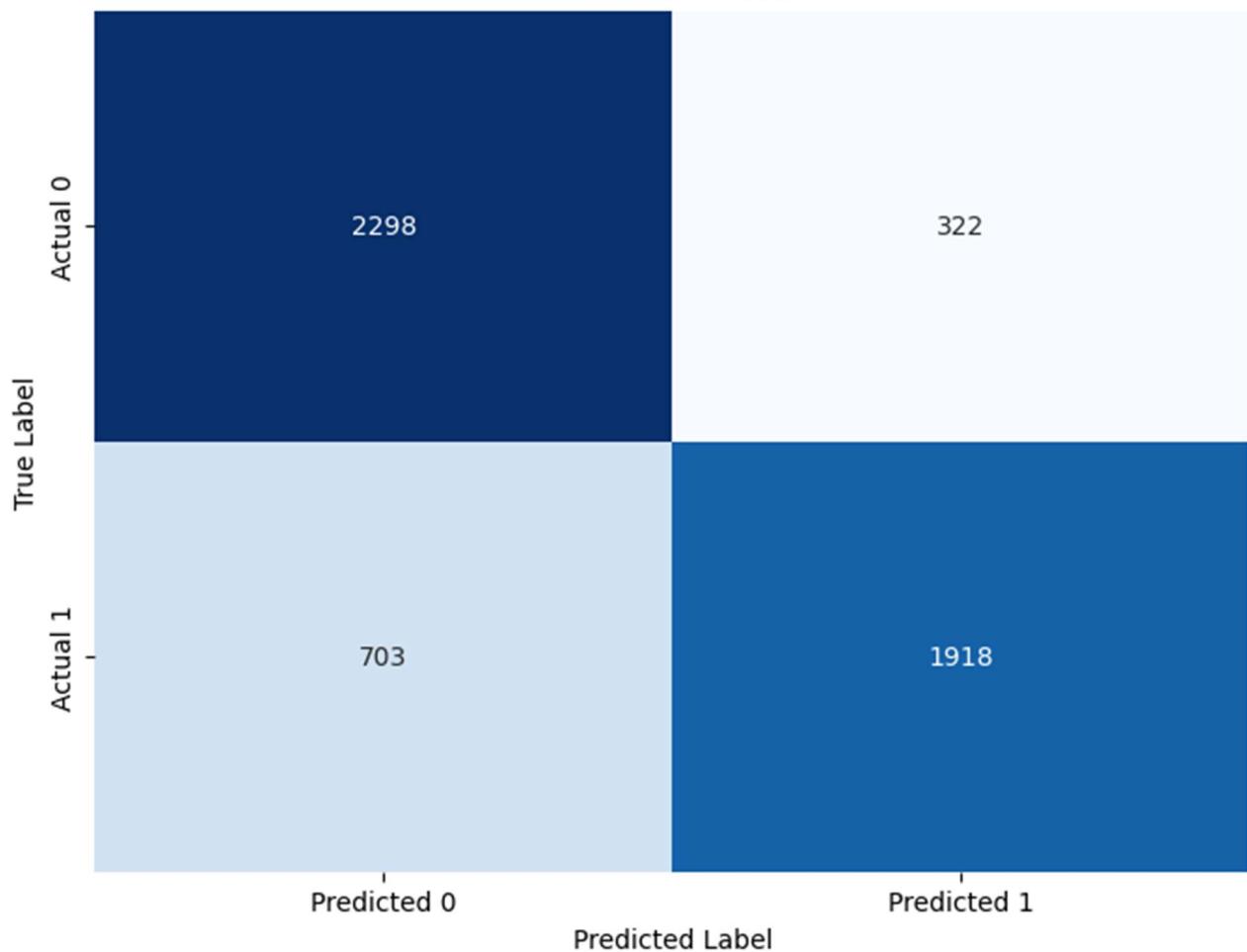
Confusion Matrix



```
Model: DecisionTreeClassifier
Accuracy test: 0.8044 | Accuracy Training: 0.8145
F1-score test: 0.7891 | F1-score Training: 0.7999
roc_AUC-score test: 0.8781 | roc_AUC-score Training: 0.8914
Differenza Accuracy: 0.0101
Differenza F1-score: 0.0107
Differenza AUC: 0.0133
Importanza delle Feature per Decision Tree:
'person_income': 0.3486
'loan_int_rate': 0.2915
'loan_amnt': 0.1348
'person_home_ownership_RENT': 0.1307
'credit_score': 0.0264
'person_home_ownership_OWN': 0.0159
'loan_intent_HOMEIMPROVEMENT': 0.0145
'loan_intent_MEDICAL': 0.0138
'loan_intent_VENTURE': 0.0068
'loan_intent_PERSONAL': 0.0063
'loan_intent_EDUCATION': 0.0059
'person_age': 0.0027
'cb_person_cred_hist_length': 0.0017
'person_education_Documentary': 2.9884e-04
AUC: 0.8781
```

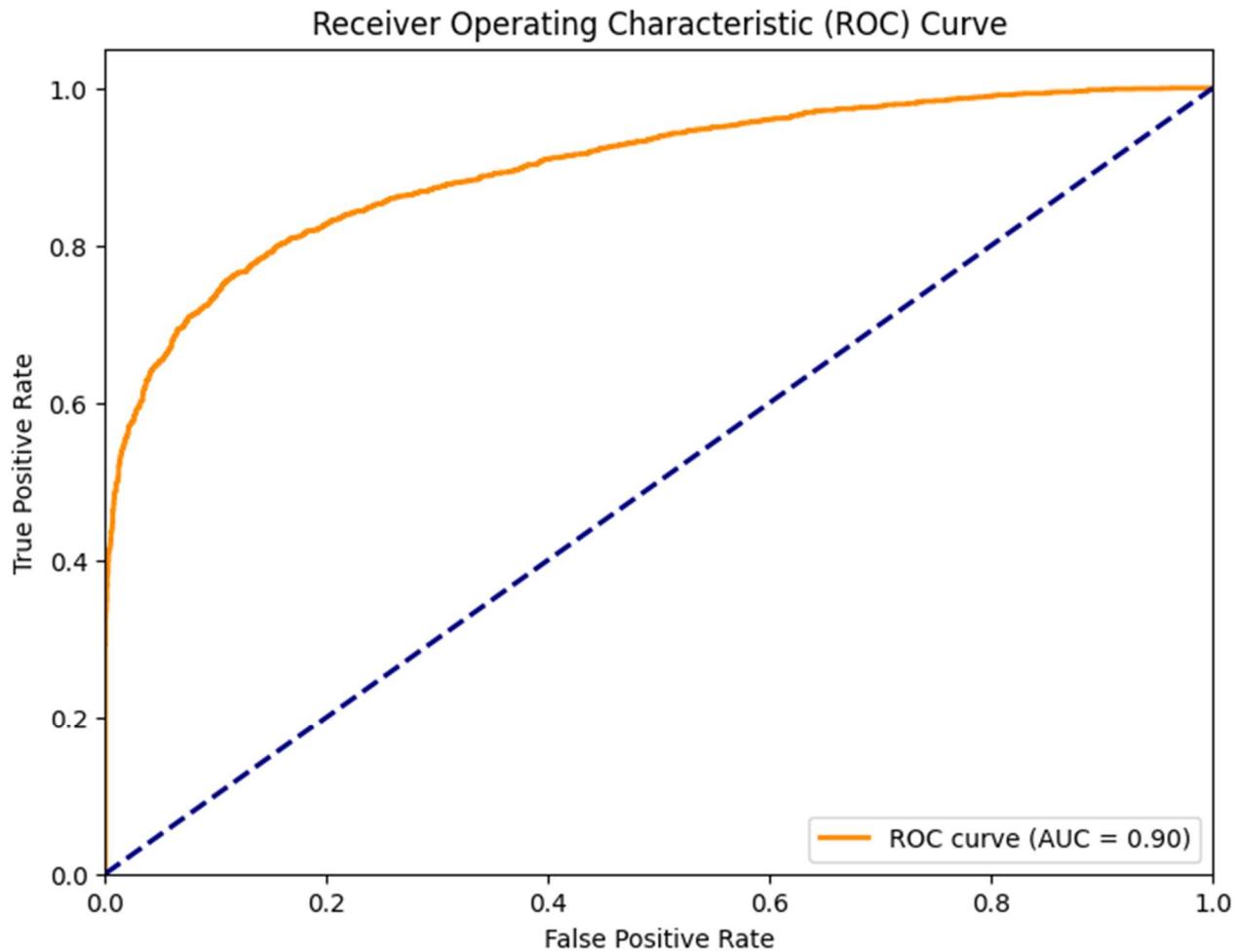


Confusion Matrix

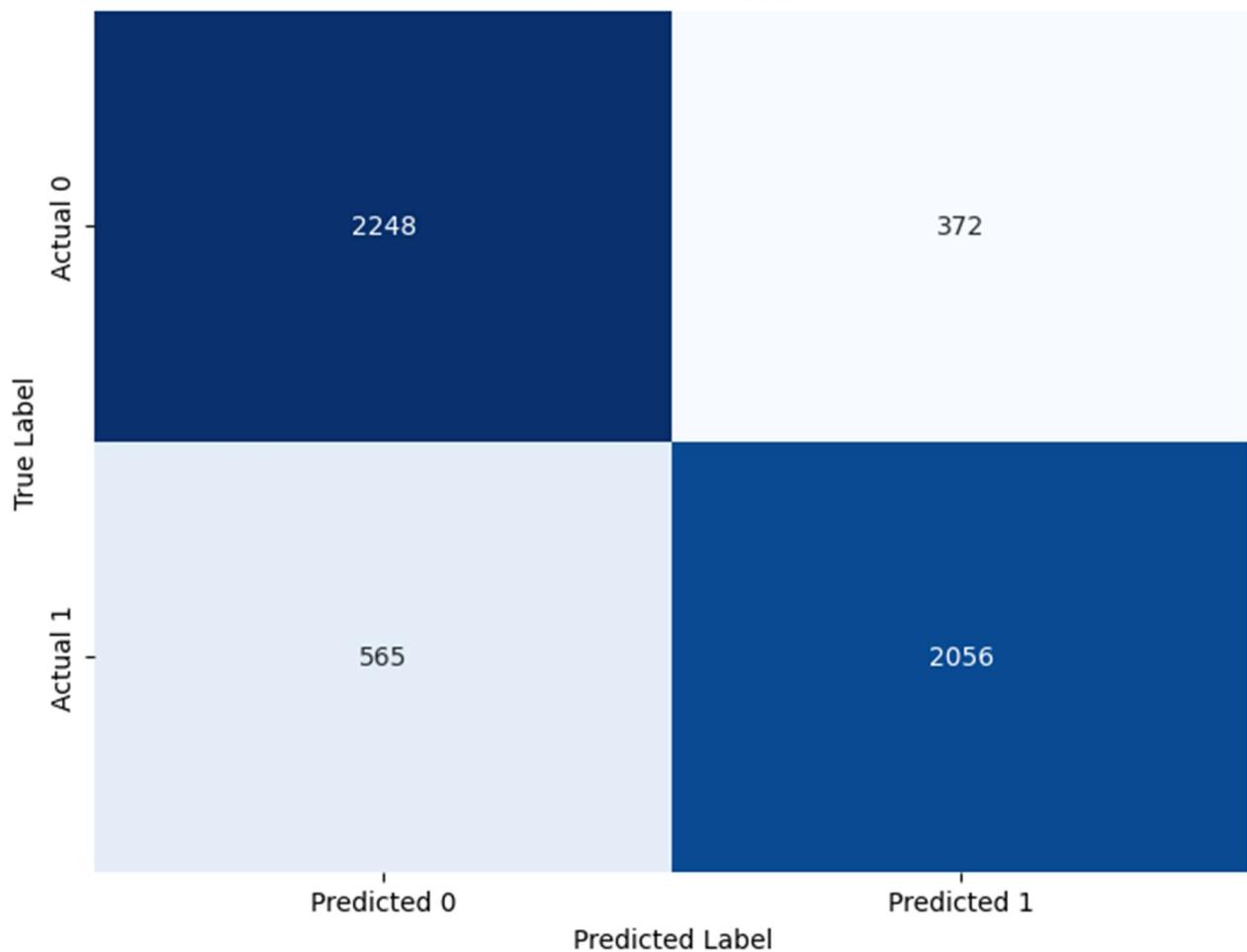


```
Model: MLPClassifier
Accuracy test: 0.8212 | Accuracy Training: 0.8376
F1-score test: 0.8144 | F1-score Training: 0.8320
roc_AUC-score test: 0.8981 | roc_AUC-score Training: 0.9252
Differenza Accuracy: 0.0164
Differenza F1-score: 0.0176
Differenza AUC: 0.0271

Importanza delle Feature (stimata con Permutation Importance) per MLP:
person_age 0.000000
person_emp_exp 0.000000
cb_person_cred_hist_length 0.000000
loan_int_rate 0.000000
person_home_ownership_OTHER 0.000000
person_home_ownership_OWN 0.000000
credit_score 0.000000
person_gender_male 0.000000
person_education_Bachelor 0.000000
person_education_Doctorate 0.000000
person_education_High School 0.000000
person_education_Master 0.000000
loan_intent_HOMEIMPROVEMENT 0.000000
loan_intent_MEDICAL 0.000000
person_home_ownership_RENT 0.000000
loan_intent_EDUCATION 0.000000
loan_intent_PERSONAL 0.000000
loan_intent_VENTURE 0.000000
person_income -0.000038
loan_amnt -0.000153
dtype: float64
AUC: 0.8981
```



Confusion Matrix



Conclusioni

Adesso analizzeremo come i vari algoritmi si sono comportati.

L'algoritmo Naive-Bayes è partito con valori infelici riguardo la classificazione dei target 0, ma accettabili negli altri parametri. Esso si è rivelato il più sensibile alle sporcature del dataset rispetto agli altri. In particolare, bisogna notare come quando “loan_int_rate” è diventata la feature più importante, sebbene i valori di accuracy, F1-score ed AUC siano peggiorati la confusion matrix ha dato risultati promettenti.

Tutti gli algoritmi si sono rivelati relativamente solidi in casi di sporcatura “realistica” (pari od inferiore al 30% del dataset) tranne nel caso della sporcatura “errori di selezione”, nella quale già al 30% di sporcatura sia il Naive-Bayes che l'albero decisionale hanno iniziato a rivelare problemi di leggero underfitting.

L'albero decisionale ed il MLPClassifier si sono rivelati particolarmente solidi anche nelle sporcature a livelli “irrealistici”.

Da notare che il MLPClassifier è stato impostato ad un solo livello nascosto e 30 neuroni, quindi aumentando la complessità della struttura probabilmente sarebbe stato capace di classificare anche in quei rari casi che ha fatto degli errori.

È bene notare che l'albero decisionale è un metodo molto simile a quello utilizzato dalle banche per decidere i prestiti, quindi la solidità di questo algoritmo potrebbe essere solo specifica per questo dataset.

Prendendo in considerazione tutte le osservazioni precedenti si può affermare che il MLPClassifier è l'algoritmo generico migliore grazie alla sua solidità rispetto agli errori introdotti ed alla sua non specificità.

Bisogna, però, fare attenzione agli errori di selezione in quanto hanno causato problemi di allenamento a valori di sporcatura molto bassi.