Università Politecnica delle Marche
Facoltà di Ingegneria
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Insegnamento: Manutenzione Preventiva per la Robotica e l'Automazione Intelligente

Denial of service:
classificazione attacchi su Modbus in sistemi di automazione

STUDENTI:

- Lorenzo Olivieri
- Martina Mammarella

ANNO ACCADEMICO:

2023/2024

SOMMARIO

INDICE DELLE FIGURE

INTRODUZIONE

Il progetto realizzato si pone come obiettivo quello di classificare diverse tipologie di attacchi DoS su sistemi automatici, a partire da un paper e da un dataset di riferimento. In particolare, si vogliono determinare le prestazioni di diversi algoritmi di classificazione in relazione a diverse durate dei tempi di attacco e dei tempi di raggruppamento (per l'estrazione delle features di interesse).

Si riportano di seguito, rispettivamente, i link per accedere al paper e al dataset introdotti in precedenza.

- https://old.cisuc.uc.pt/publication/show/5473
- https://github.com/tjcruz-dei/ICS_PCAPS?tab=readme-ov-file

Il lavoro si articola nelle seguenti fasi, che vengono introdotte nei successivi capitoli:

- Analisi: studio del paper, del dataset e degli elementi introdotti in questi ultimi;
- Progettazione: definizione generale delle scelte implementative e, quindi, del lavoro da realizzare;
- Implementazione: sviluppo del codice;
- Analisi dei risultati: studio dei risultati ottenuti.

ANALISI

Innanzitutto, è necessario definire l'ambito di lavoro, partendo quindi dall'introduzione ai concetti di sistemi ed operazioni SCADA e Cyber Physical System:

- Sistemi SCADA (Supervisory Control and Data Acquisition): sistemi che consentono la supervisione, raccolta dati, automatizzazione dei processi in ambito industriale.
- Cyber Physical System: sistemi che integrano il mondo fisico con quello digitale andando a combinare diversi componenti appartenenti a questi due mondi al fine di poter analizzare le informazioni raccolte e prendere determinate decisioni utilizzando specifici algoritmi.

A causa della scarsa disponibilità di dati per sistemi SCADA sia in normali condizioni di funzionamento che in presenza di attacchi, per la costituzione dei dataset viene sviluppato un Cyber Physical System costituito dalle seguenti componenti:

- HMI (Human Machine Interface): interfaccia che consente all'uomo di interagire con il sistema;
- PLC (Programmable Logic Controller): riceve gli input, gli elabora secondo la logica di programmazione ed invia comandi di controllo al VFD;
- RTU (Remote Terminal Unit): unità di acquisizione e trasmissione dei dati, è basata su Arduino;
- VFD (Variable Frequency Drive): dispositivo che si occupa del controllo della velocità del motore trifase variando la corrente di alimentazione;
- MOTORE TRIFASE: elemento meccanico che si occupa della realizzazione dell'azione fisica;

Per avere una visione più chiara del sistema nel suo complesso, di seguito è illustrata la configurazione logica dei suoi componenti:

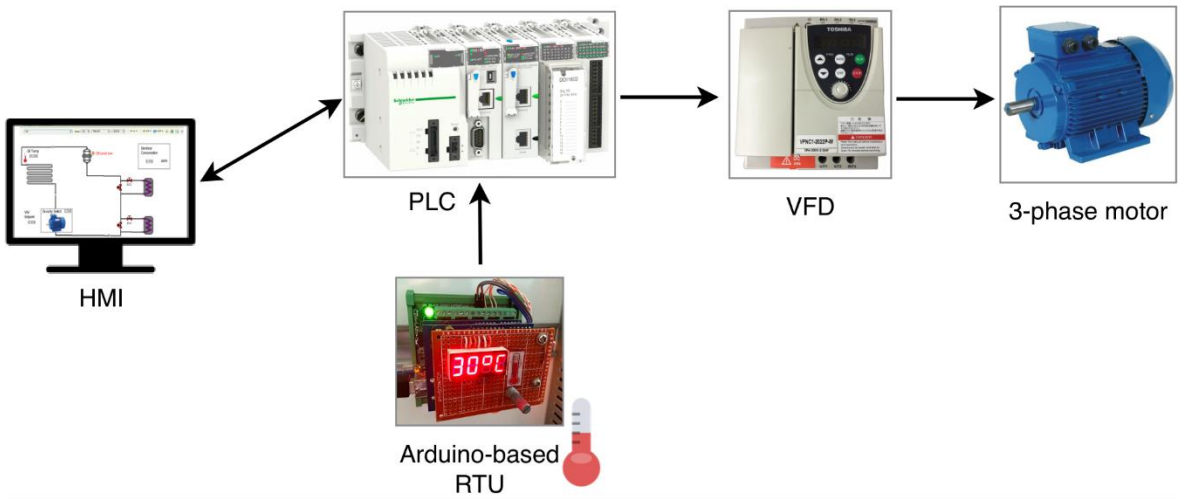

Figura 1: Schema set up utilizzato

Per quanto riguarda i protocolli su cui si basano gli attacchi, si fa riferimento a quelli che compongono la pila protocollare ISO/OSI.
A livello applicativo è possibile trovare anche il protocollo MODBUS che viene descritto, in linea generale, nel prossimo paragrafo.

MODBUS OVER TCP/IP

Protocollo utilizzato nei contesti industriali e di automazione, in particolare questa versione basata su TCP/IP consiste in un adattamento del protocollo seriale Modbus al caso di utilizzo della pila protocollare TCP/IP ereditandone le caratteristiche principali. Dunque, è fortemente utilizzato in quei casi in cui è necessaria la comunicazione su reti Ethernet di diversi dispositivi che, come nel caso in esame, devono comunicare con un sistema centrale per controllare processi in tempo reale.
Uno dei suoi principali vantaggi è quello di riuscire a connettere diversi dispositivi su una rete.

Lavora principalmente tra i livelli 5 e 7 della pila ISO/OSI, facendo affidamento sui livelli inferiori per quanto riguarda il trasporto dei dati.
Il frame è semplificato rispetto a Modbus RTU dato che non sono richiesti campi per il controllo degli errori, a livello generale può essere descritto come segue:

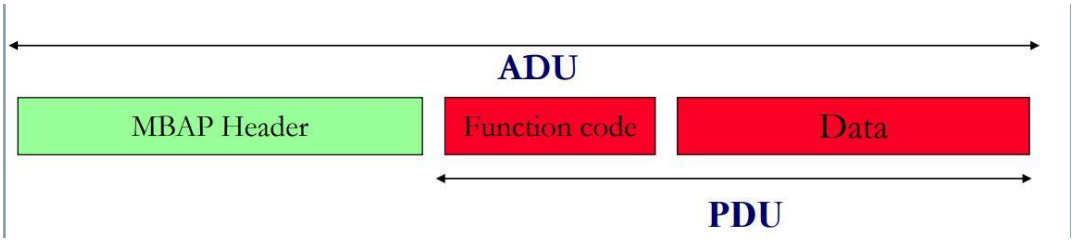

Figura 2: Frame Modbus over TCP/IP

Il paradigma di comunicazione utilizzato è quello client/server, in un'infrastruttura generale che si configura nel seguente modo:

- INTERFACCIA CLIENT MODBUS: avvia la comunicazione, invia richieste al server per leggere/scrivere dati;
- SERVER MODBUS: elabora le richieste ricevute dal client, rispondono eseguendo specifiche azioni;
- INTERFACCIA BACK END MODBUS: interfaccia che si colloca fra client e server per l'elaborazione del flusso dei dati fornisce anche supporto con altre applicazioni.

Tali sistemi sono esposti, in normali condizioni operative, a diverse tipologie di attacchi. Come da consegna ci si sofferma su attacchi di tipo Denial of Service che determinano l'inaccessibilità del servizio per utenti legittimi. Di seguito si riporta una breve descrizione delle categorie di attacchi DoS introdotti nel paper di riferimento relativamente ai quali si hanno i diversi dataset a disposizione.

- PING FLOODING: l'attaccante cerca di sovraccaricare un dispositivo di rete inviando a quest'ultimo una consistente quantità di richieste ICMP; in tal modo cerca di esaurire o limitare significativamente la capacità del sistema di gestire le risposte; (anche per quelle richieste che provengono da utenti legittimi)
- TCP SYN FLOODING: l'attaccante si insinua nel processo di handshake a tre vie che viene utilizzato dal protocollo TCP per stabilire la connessione fra client e server; nella prima fase l'attaccante invia una grande quantità di pacchetti SYN (richiesta di connessione) al server che determinano l'invio del pacchetto SYN-ACK da parte del server all'indirizzo IP indicato dall'attaccante nel primo pacchetto da esso inviato; il server non riceverà mai il pacchetto ACK da parte dell'attaccante e quindi resteranno un gran numero di connessioni incomplete che causano l'esaurimento delle sue risorse; (c'è un limite massimo di connessioni simultanee che possono essere gestite dal server)
- MODBUS QUERY FLOODING: l'attaccante invia un'enorme quantità di richieste di lettura/ scrittura in un breve intervallo di tempo ad un dispositivo che comunica per il tramite di Modbus; nel caso specifico degli Reading Holding Registers le richieste vengono inviate verso questi registri che sono solitamente utilizzati per memorizzare parametri di configurazione e una parte dei dati operativi (valori a 16 bit); si tratta di richieste indirizzate o verso specifici registri ma, nel caso più generale, queste richieste vengono indirizzate verso una vasta gamma di registri rendendoli così inutilizzabili.

IL DATASET

Il dataset fornito è costituito da tre archivi compressi, all'interno dei quali troviamo diversi file in formato PCAP (raccolta di insiemi di eventi che si susseguono sulla rete, ad ogni livello della pila TCP/IP) organizzati in cartelle. Di seguito è mostrata la loro struttura e descrizione dettagliata:

- FILE 1:
    - Cartella “clean”: nessun attacco
    - Cartella “mitm”: attacchi Man in the Middle ARP-based
    - Cartella “modbusQuery1/2”: attacchi MODBUS query flooding
    - Cartella “pingFloodDDoS”: attacchi ICMP flooding
    - Cartella “tcpSYNFloodDDoS: attacchi TCP SYN flooding
- FILE 2:
    - Cartella “modbusQueryFlooding”: attacchi Modbus query flooding
    - Cartella “pingFloodDDoS”: attacchi ICMP flooding
    - Cartella “tcpSYNFloodDDoS”: attacchi TCP SYN flooding
- FILE 3:
    - Cartella “modbusQueryFlooding”: attacchi Modbus query flooding
    - Cartella “pingFloodDDoS”: attacchi ICMP flooding
    - Cartella “tcpSYNFloodDDoS”: attacchi TCP SYN flooding

In tale contesto, infine, richiedono una riflessione approfondita le diverse features che sono utilizzate nel paper per la realizzazione dello studio.

PROGETTAZIONE

A partire da quanto descritto in precedenza, si procede con la definizione del progetto che deve essere realizzato. Si sottolinea che sono diverse le ipotesi avanzate: varie volte dalle fasi di implementazione e/o analisi dei risultati è stato necessario tornare indietro al fine di migliorare e/o raffinare sia la fase di progettazione che le stesse fasi di implementazione e/o analisi dei risultati.
Innanzitutto, si stabilisce che deve essere effettuata l'analisi contemporanea di insiemi di pacchetti interni ai singoli file: le catture, relativamente alle quali devono essere estratte una serie di features e di cui si conosce la classe di appartenenza (ricavata sfruttando le informazioni relative al tempo di inizio dell'attacco e alla durata).
Sulla base di questi elementi possono quindi essere creati diversi dataframe contenenti le catture relativamente alle quali vengono calcolate le features aggregate e a cui è possibile associare la classe. Il dataframe finale utilizzato per l'applicazione degli algoritmi deve essere costituito dalla concatenazione di tutti questi ultimi.
Si definiscono, dunque, quali sono le features aggregate da estrarre da raggruppamenti di pacchetti appartenenti alla stessa classe (tot=52 features):

- Conteggio protocolli (eth, arp, udp, tcp, icmp, ip, mbtcp), pacchetti, flag TCP (syn, ack) per cattura;
- Rapporto conteggio protocolli per numero di pacchetti nella singola aggregazione;
- Rapporto conteggio TCP syn su conteggio TCP ack;
- Deviazione standard, moda, massimo, minimo, entropia dell'inter packet arrival time;
- Deviazione standard, moda, entropia dell'indirizzo IP (src e dst), porte TCP (src e dst);

- Calcolo di quante volte gli indirizzi IP di sorgente/destinazione presenti nel raggruppamento dei pacchetti compaiono all'interno di essa e calcolo di massimo, minimo, moda, deviazione standard ed entropia;
- Moda e deviazione standard della lunghezza di tutto il layer IP (compresi quelli superiori);
- Numero di byte per unità di tempo (5 secondi);
- Conteggio di modbus request e response;
- Rapporto conteggio modbus request su modbus response;
- Conteggio ICMP echo request e echo reply;
- Rapporto echo request su echo response.

È necessario prevedere, a monte del calcolo delle features definite in precedenza, una fase di preprocessing in cui viene effettuata la pulizia delle features in cui si provvede alla:

- Conversione degli elementi esadecimali in interi decimali
- Conversione dei valori str in float
- Conversione degli indirizzi ip in interi decimali

Al fine di poter mappare i dati contenuti nei file in dataframe e per poter gestire correttamente i valori Null, si osserva che è necessario convertire i diversi file del dataset in file in formato json attraverso l'utilizzo di Wireshark.
Successivamente, per evitare di appesantire eccessivamente le diverse esecuzioni, si decide di scrivere i diversi dataframe ottenuti in forma compressa sul disco, contestualmente alla creazione di questi ultimi.
Si decide di applicare il test ANOVA (Analysis of Variance) per determinare la rilevanza delle diverse features, per eliminare quelle che non dovranno poi essere prese in considerazione, in quanto correlate a quelle in gioco.
Segue l'applicazione degli algoritmi di machine learning di classificazione

per ogni tempo di attacco e per ogni intervallo di raggruppamento definito per l'estrazione delle diverse features. In particolare, si decide che per ogni caso lo stesso modello deve essere eseguito dieci volte al fine di poter opportunamente valutare i risultati ottenuti mediante l'utilizzo delle matrici di confusione, di grafici a linee e a barre che descrivono rispettivamente l'andamento delle accuratezze, delle medie e delle deviazioni standard di quest'ultima.
Relativamente alla scelta dei vari random state da utilizzare come parametri nei modelli utilizzati si decide di voler applicare sempre gli stessi per ogni iterazione dei diversi cicli eseguiti: per fare questo occorre generare una lista statica di questi elementi che, man mano, vengono richiamati all'opportuna occorrenza.
Al fine di facilitare la lettura e l'interpretazione dei risultati, si decide per il salvataggio delle matrici di confusione e dei grafici prodotti sul disco, si definisce così una gerarchia di file e cartelle molto articolata ma con contenuti facilmente intuitivi e comprensibili.
I modelli vengono eseguiti applicando anche un'opportuna tecnica di bilanciamento: l'undersampler. A partire dai dataset, si osserva sin da subito che si ha uno sbilanciamento delle classi (attacco/non attacco) e tra le diverse alternative utili per ridurre questa problematica in fase di analisi si decide di applicare undersampler perché i risultati sono migliori: riduce gli elementi della classe maggioritaria al fine di riequilibrare gli elementi per evitare che il modello si concentri solamente su gli elementi appartenenti a quest'ultima. Per evitare che features con valori molto grandi influenzino maggiormente il modello rispetto a features con valori più piccoli, a monte dell'applicazione dei modelli bisogna applicare uno scaler che consente, appunto, di standardizzare i diversi valori: il fit però deve essere fatto solamente sui dati di training e poi si provvede a scalare sia i dati di training che di test con lo scaler definito per non alterare i risultati finali.
Al fine di migliorare la manutenibilità e la comprensibilità del codice si decide di organizzare il codice nella seguente architettura (descritta dettagliatamente nel capitolo successivo):

- File Notebook
- File Functions

IMPLEMENTAZIONE

FILE NOTEBOOK

Il notebook è organizzato in tre sezioni, individuate a partire dalla ripetizione delle stesse fasi per i file con stessi tempi di attacco (15min, 5min, 1 min). Quindi, i file elaborati nelle diverse sezioni sono i seguenti:

- SEZIONE 1:

```
clean_capture = "captures\\captures1_v2\\clean\\eth2dump-clean-0,5h_1.json"
pf_capture_1 = 'captures\\captures1_v2\\pingFloodDDoS\\eth2dump-pingFloodDDoS15m-0,5h_1.json'
pf_capture_2 = 'captures\\captures2\\pingFloodDDoS\\eth2dump-pingFloodDDoS15m-0,5h_1.json'
pf_capture_3 = 'captures\\captures3\\pingFloodDDoS\\eth2dump-pingFloodDDoS15m-0.5h_1.json'
```

- SEZIONE2:

```
pf_capture_1 = 'captures\\captures1_v2\\pingFloodDDoS\\eth2dump-pingFloodDDoS5m-0,5h_1.json'
pf_capture_2 = 'captures\\captures2\\pingFloodDDoS\\eth2dump-pingFloodDDoS5m-0,5h_1.json'
pf_capture_3='captures\\captures3\\pingFloodDDoS\\eth2dump-pingFloodDDoS5m-0,5h_1.json'
```

- SEZIONE3:

```
pf_capture_1 = 'captures\\captures1_v2\\pingFloodDDoS\\eth2dump-pingFloodDDoS1m-0,5h_1.json'
pf_capture_2 = 'captures\\captures2\\pingFloodDDoS\\eth2dump-pingFloodDDoS1m-0,5h_1.json'
pf_capture_3='captures\\captures3\\pingFloodDDoS\\eth2dump-pingFloodDDoS1m-0,5h_1.json'
```

La descrizione che segue fa riferimento ad una singola sezione, cambiando i file (tempo di attacco) è possibile applicare lo stesso ragionamento per tutte le fasi e dunque ottenere così la spiegazione dell'intero codice implementato.

Si noti che, solamente nella prima sezione viene definito il file clean_capture che contiene il dataset con le catture pulite per quel determinato tempo di cattura utilizzato nel resto del codice.
Innanzitutto, occorre effettuare l'importazione delle librerie e dei moduli utilizzati:

- pandas: consente di manipolare strutture dati tabellari
- json: consente di manipolare dati in formato.json
- numpy: consente di manipolare array
- warnings: consente di ignorare i warning non necessari
- datetime.timedelta: consente di manipolare intervalli temporali
- os: consente di interagire con il file system
- sklearn.tree.DecisionTreeClassifier: consente di implementare modelli di classificazione basati su alberi decisionali
- sklearn.model_selection.train_test_split: consente di dividere i dati in dati di training e dati di test
- sklearn.ensemble.RandomForestClassifier: consente di implementare modelli di classificazione basati su random forests
- sklearn.svm.SVC: consente di implementare modelli di classificazione basati su Support Classifier
- sklearn.preprocessing.StandardScaler: consente di standardizzare i dati
- sklearn.neighbors.KNeighborsClassifier: consente di implementare modelli di classificazione basati su K-NN
- sklearn.metrics.accuracy_score: consente di calcolare l'accuratezza del modello
- sklearn.feature_selection.f_classif: consente di selezionare le features utili ai fini della classificazione
- imblearn.under_sampling.RandomUnderSampler: consente di effettuare il bilanciamento delle classi basato su Undersampler
- sklearn.metrics.confusion_matrix: consente la generazione della matrice di confusione

- seaborn: consente di effettuare analisi statistiche avanzate
- matplotlib.pyplot: consente di creare grafici e visualizzazioni

Viene definito il time_aggregation che deve essere posto uguale al numero di secondi in base al quale i pacchetti vengono raggruppati, utilizzato quindi dalla funzione di aggregazione che verrà descritta successivamente.
Se i quattro file, per i quali viene definito il percorso come mostrato in precedenza, non sono già stati letti e poi salvati in passato e dunque estratte anche le relative features appartenenti ai livelli di interesse del pacchetto definiti in level_of_interest, occorre andare a creare i diversi dataframe che poi verranno utilizzati in futuro. A tal fine, si procede nel seguente modo ma, trattandosi per l'appunto di quattro dataframe differenti per lo stesso tempo di attacco e di cattura, ogni singolo passaggio deve essere ripetuto per ognuna delle diverse casistiche:

- Viene richiamata la funzione load_packets_from_file (passando come argomento clean_capture che contiene il percorso del file da leggere) del modulo fn che verrà introdotto successivamente; il risultato viene assegnato a packets_cc
- Viene richiamata la funzione extract_features_from_packets del modulo fn (argomenti: packets_cc e level_of_interest), le features estratte vengono assegnate alla variabile features_cc
- pd.DataFrame (features_cc): converte le features estratte in precedenza nel formato dataframe e le salva in df_cc
- features_cc.clear(): libera la memoria, avendo il dataframe mantenuto in df_cc
- df_cc.to_pickle (dfclean_disk, compression='gzip'): salva df_cc nel file definito in df_clean_disk (vedi qualche rigo precedente) usando la compressione gzip.

- Se la condizione è falsa, si utilizza la funzione read_pickle con argomenti il file in cui è stato salvato il dataframe sul disco e il tipo di compressione utilizzato (gzip); il dataframe viene caricato in df_cc.

Quindi, andando avanti nell'analisi del codice, per ogni dataframe acquisito, si procede come segue. Come fatto in precedenza, si considera solamente un dataframe per la spiegazione; anche qui cambiando i nomi alle variabili e ripetendo il ragionamento applicato al singolo caso si può ottenere la spiegazione dell'intero codice (si utilizza il dataframe associato al primo dataset con catture con attacchi all'interno, qualora necessario viene introdotto anche il dataframe associato al file con catture clean).

- Richiamo della funzione calcolo features_binarie del modulo fn, passando come parametro df.
- Conversione della colonna frame.frame.time_utc in formato datetime utilizzando pd.to_datetime (argomento: df['frame.frame.time_utc'])
- Ordinamento delle righe del dataframe sulla base dei valori della colonna frame.frame.time_utc, utilizzando la funzione df_cc.sort_values (argomento: by='frame.frame.time_utc')
- Salvataggio della features frame.frame.time_utc della prima riga del dataframe df (iloc [0]) e del tempo di inizio e di fine dell'attacco in 3 diverse variabili (start_time, end_time, end_time2) al fine di suddividere il dataframe in 3 parti: clean-attack-clean
- Suddivisione del dataframe sulla base di quanto individuato nel punto precedente; quindi, vengono creati i seguenti dataframe con condizioni legate ai tempi individuati e con l'utilizzo della funzione copy () che consente di creare una copia indipendente del subset di righe selezionate per ogni casistica:
    - df_cc1
    - df_attack
    - df_cc2
- Calcolo delle features aggregate sui singoli dataframe ottenuti; dunque, si richiama la funzione calcola_features del modulo fn su

ogni dataframe ottenuto dal punto precedente specificandolo come parametro per ogni diversa chiamata insieme al time_aggregation, ottenendo i seguenti dataframe aggregati
    - df_cc1_aggregation
    - df_cc2_aggregation
    - df_att_aggregation
- Assegnazione della classe ai singoli dataframe aggregati: 0 clean, 1 attack: tale informazione viene memorizzata nella colonna label
  NB: Il dataframe associato al file clean non viene suddiviso dato che contiene solamente dati clean e quindi viene richiamata la funzione calcola_features su quest'ultimo e poi al dataframe aggregato che si ottiene viene associata la classe 0
- Concatenazione di tutti i dataframe ottenuti, usando pd.concat(array con i nomi dei dataframe da concatenare). Il nuovo dataframe viene assegnato a df_tot.
- Conversione dei valori contenuti nelle colonne di df_tot ipat_std, ipat_max, ipat_min, ipat_mode da timedelta in secondi, utilizzando della funzione df.total_seconds ()
- Sostituzione dei valori Nan di df_tot con 0, utilizzando fillna
- Separazione delle features dalla variabili target di df_tot (contenute rispettivamente in X e y)
- Applicazione del test Anova su df_tot (si utilizzata codice predefinito adattato per il test)
- Utilizzando X.drop si provvede all'eliminazione delle colonne non rilevanti individuate, salvate precedentemente in features_to_discard; le modifiche vengono effettuate direttamente sull'oggetto originale dato che si utilizza l'argomento inplace=True.
- Applicazione degli algoritmi di classificazione (10 volte) e costruzione dei grafici:
    - Vengono definiti 4 array per la memorizzazione delle accuratezze che verranno poi calcolate per ogni modello utilizzato durante ogni iterazione: accuracies_dt, accuracies_rf, accuracies_knn, accuracies_svc. Inoltre si provvede ad

assegnare a randomstate_array il risultato dell'attivazione di init_load_rstate ()

- Definizione del ciclo for per l'esecuzione delle 10 iterazioni
- Si utilizza la funzione train_test_split per suddividere il dataset in 2 parti: 70% dati di train memorizzati in X_train, 30% dati di test memorizzati in X_test mentre in y_train si hanno le etichette associate ai dati di train e in y_test si hanno le etichette associate ai dati di test. Si utilizzata come random_state un seme che sarà diverso (casuale) per ogni iterazione ma riproducibile in quanto è un elemento del randomstate_array, individuato quindi nella lista a partire da un contatore associato al for stesso
- Standardizzazione: dopo aver creato l'istanza scaler dello standardizzatore StandardScaler si utilizza scaler.fit_transform(X_train) per calcolare la media e la deviazione standard per ogni colonna usando i dati di training, poi viene applicata la standardizzazione a questi ultimi e i risultati vengono memorizzati in X_train_scaled. Gli ultimi 2 passaggi vengono ripetuti anche sui dati di test ma utilizzando i parametri calcolati su dati di training, quanto ottenuto viene salvato in X_test_scaled
- Addestramento del modello Decision Tree utilizzando i risultati ottenuti in precedenza, effettua quindi previsioni sui dati di test, utilizzando codice predefinito. In accuracy_dt si memorizza l'accuratezza per quell'iterazione che poi confluirà nella lista accuracies_dt attraverso l'utilizzo della funzione append a cui viene passato come argomento accuracy_dt.
- Analogamente, adattando il codice alla specifica casistica, si procede per i modelli: Random Forest, KNN, SVC
- Viene definito il dizionario iterations_name per associare i nomi dei modelli di machine learning alle rispettive matrici di confusioni che vengono generate dalle diverse iterazioni, sui dati opportuni

    - Qualora non esista la directory per la memorizzazione dei risultati ottenuti per la specifica casistica, viene creata con la funzione makedirs
    - Utilizzando codice predefinito per la creazione dei grafici ed adattandolo poi alla specifica casistica viene creata la heatmap e salvata contestualmente alla directory corrente
    - Per ciascun modello si provvede al calcolo della media e delle deviazioni standard delle accuratezze attraverso l'utilizzo delle apposite funzioni np.mean e np.std passando come argomenti le liste ottenute; i risultati vengono memorizzati in variabili del tipo mean_accuracy_dt e std_accuracy_dt
    - Viene generato un grafico a barre orizzontali, sempre adattando il codice predefinito alle specifiche esigenze, che confronta medie delle accuratezze e deviazioni standard dei modelli. Sulle ascisse in alto troviamo i valori della deviazione standard (arancio), in basso i valori delle medie(blu) e sulle ordinate i nomi dei diversi modelli.
    - NB: Un analogo ragionamento viene applicato anche nel caso dell'utilizzo dell'undersampling ma, in tal caso, inizialmente viene definito l'undersampler istanziandolo a partire da RandomUnderSampler che è una tecnica di bilanciamento che riduce le dimensioni della classe maggioritaria in modo casuale ma, siccome vogliamo un sottocampionamento riproducibile nelle altre casistiche, il random_state all'iterazione i viene letto a partire dalla corrispondente posizione nella lista dei random_state. X_train_balanced e y_train balanced si ottengono applicando fit_resample agli elementi passati come argomenti (X_train_scaled e y_train)
- Infine, viene generato un grafico a linee che consente di avere una visione panoramica e quindi di confrontare la media delle accuratezze dei 4 modelli (sia nel caso senza che nel caso di utilizzo dell'undersampler), procedendo sempre in maniera analoga ai casi

precedenti. Sulle ascisse si hanno i tempi di cattura e sulle ordinate i valori delle medie delle accuratezze.

FUNCTION_NOTEBOOK

Anche in tal caso si procede con l'importazione di moduli e librerie necessarie per lo sviluppo del codice:

- Json: consente la manipolazione dei dati in formato json
- pandas: consente la manipolazione di dati strutturati
- numpy: consente di manipolare array multidimensionali
- ipaddress: consente di manipolare indirizzi id
- from scipy.stats import entropy: consente di manipolare l'entropia delle distribuzioni di probabilità
- pickle: consente la serializzazione/deserializzazione di oggetti Python
- os: consente di interagire con file system
- random: consente di generare numeri casuali

INIT_LOAD_RSTATE

Funzione che consente di inizializzare o di caricare un array di numeri casuali da un file salvato sul disco. L'introduzione dell'if consente di controllare, attraverso l'utilizzo di os.path.exists che il file specificato come argomento randomstate_array.pkl non esiste nella directory corrente. Se la condizione è vera allora viene creato randomstate_array che contiene 10 numeri casuali a 32bit (il singolo numero viene generato da random.getrandbits(32)), si provvede alla stampa di quest'ultimo e poi utilizzando la funzione open si apre il file randomstate_array.pkl specificato come argomento in scrittura binaria ed utilizzando with viene garantito che alla fine delle operazioni il file viene chiuso in maniera automatica; pickle.dump(random_array, file) salva l'array passato come primo

parametro nel file effettuandone la serializzazione; infine randomstate_array viene ritornato al chiamante.
Se, invece, la condizione introdotta dall'if è falsa si utilizza sempre la open per aprire il file randomstate_array.pkl in modalità lettura binaria e quindi per leggere il contenuto dell'array ivi contenuto. In randomstate_array si memorizza il risultato di pickle.load(file) che effettua la deserializzazione del file binario; anche in tal caso quanto ottenuto viene restituito al chiamante.

LOAD_PACKETS_FROM_FILE

Funzione che consente di caricare e leggere dati in formato json da un file specificato restituendoli come oggetti del linguaggio. Ha un unico parametro: file_path che specifica quale file, in quale percorso deve essere letto; attraverso l'utilizzo della open (file_path, rb) il file viene aperto in modalità lettura binaria, anche in tal caso si ricorre all'utilizzo di with sulla base di quanto illustrato in precedenza; json.load (file) interpreta il contenuto del file specificato, lo converte in un oggetto Python ed assegna il risultato a data che viene restituito al chiamante.

FLATTEND_DICT

Funzione che consente di trasformare un dizionario annidato in una struttura piatta; i parametri sono: d ovvero il dizionario e parent_key che è una stringa opzionale, una sorta di prefisso delle chiavi del dizionario, usato nella costruzione. Dunque, si procede come segue:

- inizializzazione della lista vuota iteams che conterrà i risultati
- controlla se d è un dizionario
- Per ogni elemento del dizionario:

    - se esiste parent_key, in caso di chiamata ricorsiva sugli elementi del tipo dizionario, la nuova chiave viene costruita concatenando parent_key e la chiave corrente k, altrimenti la chiave sarà costituita solo da k
    - qualora l'elemento considerato sia un altro dizionario si richiama ricorsivamente flatten_dict passando la new_key come parent_key; qualora sia una lista allora si scorre ogni elemento della lista per gestire opportunamente con chiamata ricorsiva il singolo elemento (che potrà essere dei 3 tipi in esame)
    - se il valore corrente è un elemento semplice allora la coppia new_key v (chiave-valore) viene aggiunta ad items
- se il valore considerato è un elemento semplice fin dall'inizio, aggiunge la coppia new_key-valore ad iteams
- ritorna iteams al chiamante

EXTRACT_FEATURES_FROM_PACKETS

Funzione che consente di estratte ed "appiattire" features dei levels_of_interest dei packets, questi ultimi sono argomenti della funzione. Nel dettaglio:

- viene definita all_features, l'array da restituire al chiamante
- per ogni packet in packets:
    - Attraverso l'utilizzo della get si cerca la chiave source in packet, se esiste viene restituito il valore associato altrimenti un dizionario vuoto, la chiave layers viene ricercata all'interno di quanto restituito in precedenza; quindi, se tale elemento esiste viene restituito il valore associato altrimenti viene restituito un dizionario vuoto
    - Viene analizzato ogni livello del pacchetto (definito da coppie layer_name e layer_content) e se quel livello appartiene a levels_of_interest allora i dati contenuti in layer_content e

layer_name vengono “appiattiti” attraverso l’utilizzo della flatten_dict e poi aggiunti al dizionario features
    - Features (per il singolo) ottenuto viene aggiunto ad all_features
- La lista dei dizionari ottenuta viene restituita al chiamante

CALCOLO_FEATURES_BINARIE

Funzione che aggiunge al dataframe passato come argomento le features binarie (1/0) al fine di rappresentare meglio le informazioni in input per l’elaborazione successiva

- divide ogni stringa nella colonna frame.frame.procols in una lista di protocolli che assegna alla colonna frame.frame.split, utilizzando str.split con argomento: sulla base del quale avviene la suddivisione
- per ogni lista della colonna protocols_split e su ogni protocollo in quelle liste estrae tutti i protocolli mettendoli in un'unica lista, elimina poi i duplicati utilizzando set ed assegna gli elementi ottenuti a protocols
- per ogni protocollo unico trovato aggiunge una nuova colonna nel dataframe: se il protocollo è presente in protocols_split di quel pacchetto considerato allora assegna il valore 1 altrimenti 0
- elimina la colonna temporanea protocols_split e, se presente, ethertype

CALCOLA_FEATURES

Funzione che consente il calcolo delle diverse features da utilizzare per lo studio.
Accetta come parametri di input il dataframe relativamente al quale devono essere calcolate le features e _time_groupby che rappresenta la finestra temporale su cui fare l’aggregazione dei pacchetti. Quindi restituisce un dataframe con le features aggregate calcolate sulle catture.

Innanzitutto, provvede alla conversione del campo frame.frame.frame_utc in formato datetime al fine di consentire il raggruppamento temporale.
Viene creato un nuovo dataframe vuoto df_aggregation che conterrà tutte le features calcolate, aggregate.
Si calcola la quantità di pacchetti nei quali sono presenti i protocolli: eth, ip, arp, udp, tcp, mbtcp, icmp. (La logica è sempre la stessa, per tutti i protocolli, pertanto si analizza solamente il caso eth).
La groupby su dataframe provvede al raggruppamento dei diversi pacchetti ivi contenuti sulla base di _time_group_by facendo riferimento ai valori nella colonna frame.frame.time_utc e sui risultati ottenuti applica come lambda function sull'elemento x sum() che consente di calcolare il numero di pacchetti per i quali la features eth selezionata è uguale a 1 (vedi calcolo features_binarie); reset_index() trasforma il risultato della funzione di aggregazione in un dataframe e rinomina la colonna che contiene i conteggi con il nome count, solo quest'ultima viene presa in considerazione per l'assegnazione a eth_count.
NB: Nel caso dei protocolli arp, udp, icmp deve essere verificata la presenza del nome del protocollo nelle colonne del dataframe, se non c'è allora nelle colonne arp_count/udp_count/icmp_count si ha il valore 0.
Un ragionamento analogo al precedente viene applicato per il calcolo del numero totale di pacchetti presenti nelle diverse catture che costituiscono il dataframe. Viene attivata, come visto anche in precedenza, la group_by su dataframe, si seleziona la colonna frame.frame.time_utc sul risultato e su quest'ultima viene attivata la count per il calcolo delle righe in ogni gruppo creato, i risultati ottenuti vengono assegnati a df_aggregation['pkt_count'].
Per ognuno dei protocolli introdotti in precedenza, si calcola la percentuale di pacchetti con quel protocollo per ogni cattura; quindi, tornando a descrivere solo quanto accade per eth abbiamo che df_aggregation['per_eth_count] contiene il rapporto di df_aggregation['eth_count'] /df_aggregation['pkt_count'].
Per le colonne tcp.tcp.flags_tree.tcp.flags.syn e tcp.tcp.flags_tree.tcp.flags.ack si sostituisce ai valori Nan 0 e vengono

effettuate le conversioni dei valori delle colonne da stringhe a numeri interi.
Si procede per la prima features introdotta con una nuova attivazione della group_by su dataframe, sul risultato si prende la colonna tcp.tcp.flags_tree.tcp.flags.syn e si applica la funzione lambda che provvede al conteggio degli elementi che hanno tale features =1, il risultato viene poi assegnato a df_aggregation['tcp_syn_count'].
Analogamente viene fatto per la seconda delle features in oggetto e il risultato viene assegnato a df_aggregation['tcp_ack_count']. In df_aggregation['tcp_synack_fraction'] si ha il risultato del rapporto fra df_aggregation['tcp_syn_count'] e df_aggregation['tcp_acl_count'] che, nel caso in cui df_aggregation['tcp_ack_count'] =0 è uguale a 0.
Diff()consente di calcolare la distanza temporale che intercorre fra due pacchetti, sulla base dei valori contenuti nella colonna frame.frame.time_utc di dataframe; il risultato ottenuto viene assegnato a dataframe['inter.packet_arrival_time'].
Per il calcolo delle features aggregate che si basano sull'utilizzo di inter.packet_arrival_time si richiama ancora la group_by e sul risultato che si ottiene si seleziona la colonna inter.packet_arrival_time e quindi sui vari gruppi ottenuti si applicano come lambda function la std() per il calcolo della deviazione standard la mode() per il calcolo della moda, max() per il calcolo del massimo e min() per il calcolo del minimo, entropy(x.value_counts(), len (x)) per il calcolo dell'entropia dove value_counts conta le occorrenze di ciascun valore univoco in x e len(x) è la lunghezza totale della serie; i risultati poi vengono assegnati a df_aggregation['ipat_std'],….
Sostituisce tutti i valori Nan nella colonna ip.ip.dst con ip predefinito 169.254.0.0 e attraverso una lambda function tutti i valori della colonna vengono trasformati in interi, il risultato viene assegnato a dataframe['ip.ip.dst_asint']
Anche qui si applica la group_by su dataframe e poi sulla colonna ip.ip.dst_asint del risultato si applicano le lambda function con argomenti

x.std, x.mode e entropy; i risultati vengono assegnati rispettivamente a df_aggregation[‘ip_dst_std’],[‘ip_dst_mode’] e [‘ip_dst_entropy’].
Un ragionamento analogo viene seguito anche per il calcolo di df_aggregation[‘tcp_dstport_std’], df_aggregation[‘tcp_dst_mode’],df_aggregation[‘tcp_dstport_entropy’] e per il calcolo di df_aggregation[‘tcp_srcport_std’],df_aggregation[‘tcp_srcport_mode’],[src_dstport_entropy’]. Si assegna a tcp.dstport_mbtcp il risultato che si ottiene dal filtraggio su x==502 (associata al protocollo mbtcp).
Si richiama nuovamente la group_by su dataframe ma in questo caso raggruppiamo sia sulla base di frame.frame.time_utc suddividendolo in intervalli temporali di _time_group_by secondi ma questi dati vengono poi ulteriormente raggruppati per valori univoci di ip.ip.dst_asint; size() consente di calcolare il numero di elementi presenti nei gruppi definiti dai due criteri, resta invariato il funzionamento di reset_index e il risultato viene assegnato a df_cc_1. Analogamente a quanto fatto in precedenza su df_cc_1 si richiama la group_by con lambda function min max mode entropy per il calcolo delle features aggregate sulle catture; i risultati vengono assegnati a df_aggregation[‘maxnum_unique_ipdst’],…
Sempre attivando la group_by ma in tal caso andando a selezionare rispettivamente le colonne ip.ip.src e ip.ip.dst del risultato e applicando nunique() si calcola il numero di indirizzi ip di origine e destinazione univoci presenti nelle catture; i risultati vengono assegnati a df_aggregation[‘num_unique_ipsrc’] e df_aggregation[‘num_unique_ipdst’].
Per la colonna ip.ip.len del dataframe si sostituiscono i valori Nan con 0 poi trasformano in interi e si attiva la group_by sul dataframe , sul risultato si seleziona la colonna ip.ip.len e si applicano come lambda function std, entropy, come fatto precedentemente; I risultati vengono assegnati rispettivamente a std_iplen e entropy_iplen. Si assegna a df_aggregation[‘bytes_per_timeunit’] il risultato dell’attivazione della group_by di cui si seleziona sempre la colonna ip.ip.ip per poi usare come

labda function x.sum()/_time_groupby per il calcolo del numero di bytes per unità di tempo.
Si calcola il numero di pacchetti per unità di tempo come il rapporto fra quanto contenuto nella colonna pkt_count e _time_group_by.
Per la colonna modbus.modbus.func_code vengono sostituiti i valori Nan con 67, poi viene effettuata la conversione dei valori della colonna in interi e ciò viene fatto anche per i valori della colonna frame.frame.len.
In seguito, si selezionano solamente i pacchetti modbus con codice della funzione 3 e i pacchetti di lunghezza uguale ad 85, si richiama su questi elementi filtrati la group_by e sulla colonna frame.frame.len si applica come labda function count() per il conteggio di tali elementi nei gruppi che verranno assegnati alla colonne modbus_response_count; si procede ugualmente per i pacchetti modbus con codice della funzione 3 e di lunghezza uguale a 66 ma il risultato finale viene assegnato a modbus_request_count. Si effettua il rapporto fra le ultime due variabili introdotte ed il risultato viene assegnato alla modb_req_resp_fraction di df_aggregation.
Viene introdotto un if che consente di verificare se la colonna icmp.icmp.code è presente nel dataframe.
Se la condizione è falsa allora le colonne di df_aggregation icmp_request_count, icmp_response_count e icmp_resp_fraction avranno come valori 0.
Se, invece, la condizione è vera vengono gestiti i valori Nan della colonna icmp.icmp.code a cui viene assegnato il valore 99 e poi i valori della colonna vengono trasformati in interi.
Analogamente a quanto fatto per mbtcp, si utilizza la group_by con precedente filtraggio sia per il calcolo delle richieste e delle risposte icmp nelle catture individuate tramite le group_by: si selezionano a monte solo i pacchetti icmp con codice 0 oppure 8 e poi i risultati vengono assegnati a icmp_response_count e icmp_request_count di df_aggregation. Eventuali Nan individuati nella colonna icmp_request_count vengono associati a 0. Se anche la colonna icmp_response_count di df_aggregation contiene valori Nan allora anche questi vengono associati a 0; si filtrano le righe di

df_aggregation in cui icmp_response_count!=0 e si calcola il rapporto fra icmp_request_count e icmp_response_count, ai valori Nan della colonna icmp_req_resp_fraction si associa lo 0; nel caso in cui la condizione è falsa si calcola direttamente il rapporto descritto in precedenza e si assegna il risultato a df_aggregation[‘icmp_req_resp_fraction’].
Df_aggregation viene restituito al chiamante.

ANALISI DEI RISULTATI

Di seguito si mostrano i risultati degli esperimenti condotti secondo la metodologia illustrata nei precedenti paragrafi.
Per praticità nella corrente relazione si riporteranno, al fine di mantenere il lavoro più conciso, soltanto i risultati ottenuti sottoponendo ai classificatori le catture della durata complessiva di trenta minuti considerando i diversi tempi di attacco basato su ping flooding (15m, 5m, 1m) e il caso specifico in cui il tempo di aggregazione dei vari pacchetti è pari a cinque secondi.
Si ricorda che al fine di misurare la stabilità dei modelli su diverse esecuzioni randomizzando le componenti in gioco (dataset split, undersamplers, classificatori) si è inserita la fase di addestramento e test dei modelli all'interno di un ciclo che si ripete dieci volte. Per ogni modello vengono quindi prodotte dieci matrici di confusione e poi un grafico che riporta deviazioni standard e medie delle accuratezza rispettivamente ai vari modelli utilizzati.

CATTURA PING FLOODING 15M ATTACK - AGGREGATION 5S - CON UNDERSAMPLER

Nel caso in questione i modelli DT, RF, KNN e SVC vengono addestrati su di un dataset contenente catture malevole con attacchi ping flooding della durata di 15m, che viene bilanciato per il tramite del componente undersampler.
Di seguito sono illustrate le matrici di confusione prodotte nelle dieci iterazioni:

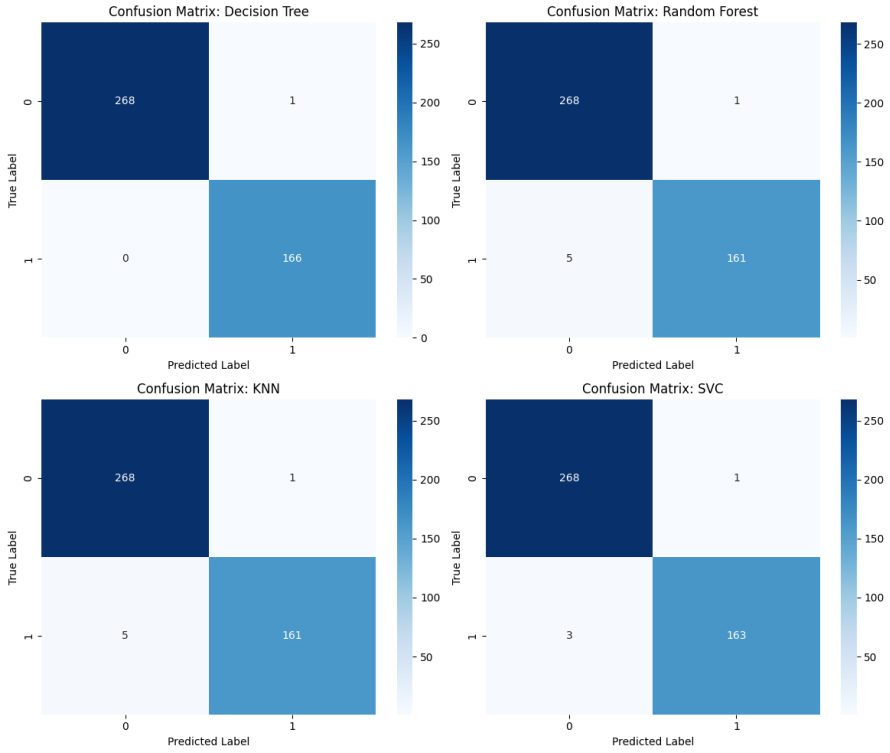

Figura 3 - Matrice di confusione - 15m attack - aggregation 5s - i=0 – undersampler

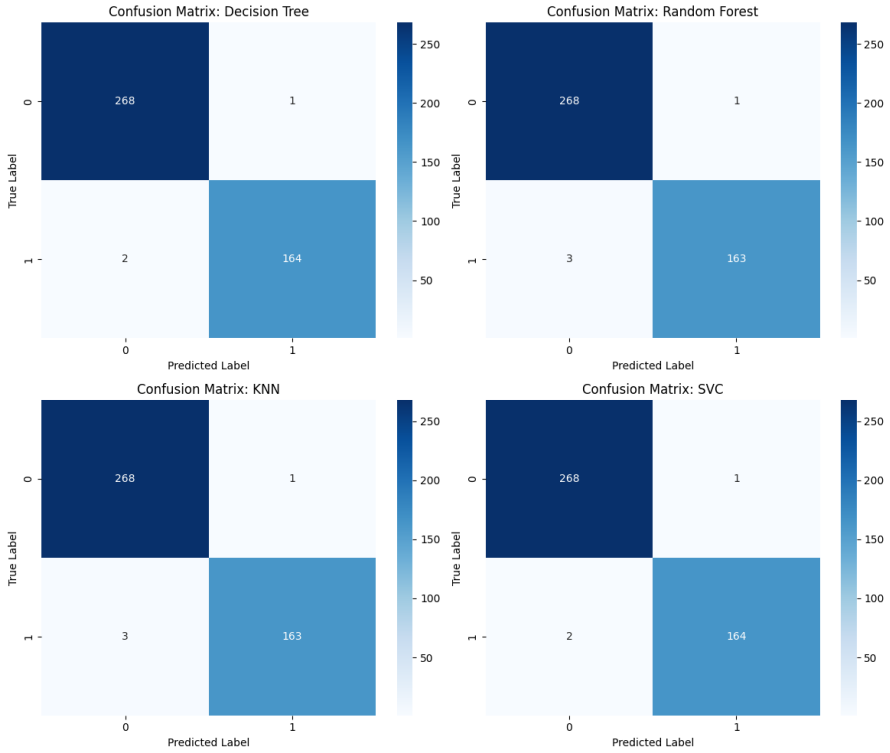

Figura 4 - Matrice di confusione - 15m attack - aggregation 5s - i=1 - undersampler

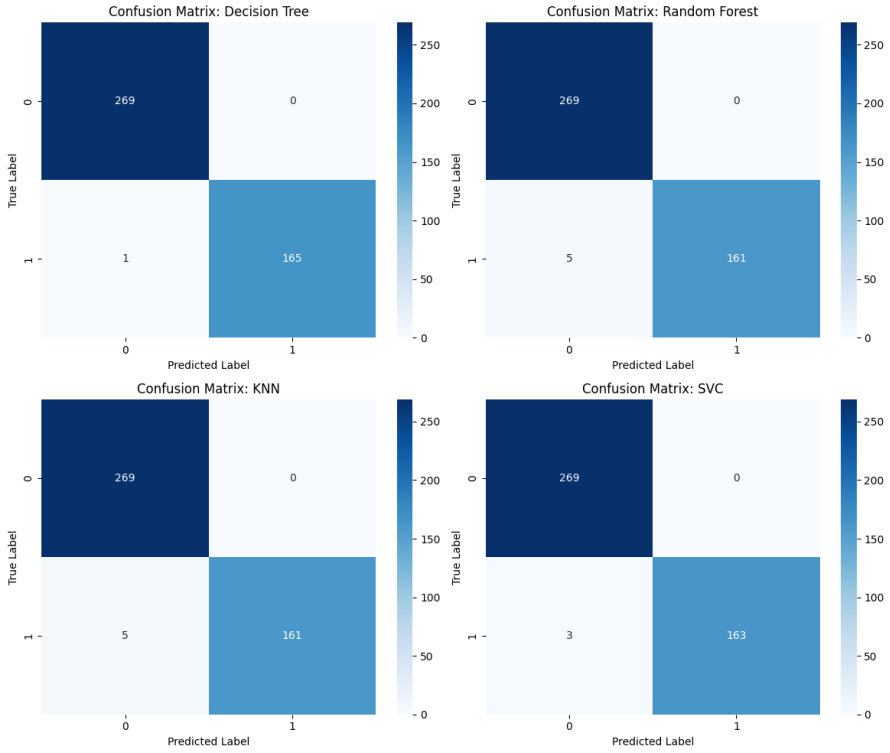

Figura 5 - Matrice di confusione - 15m attack - aggregation 5s - i=2 - undersampler

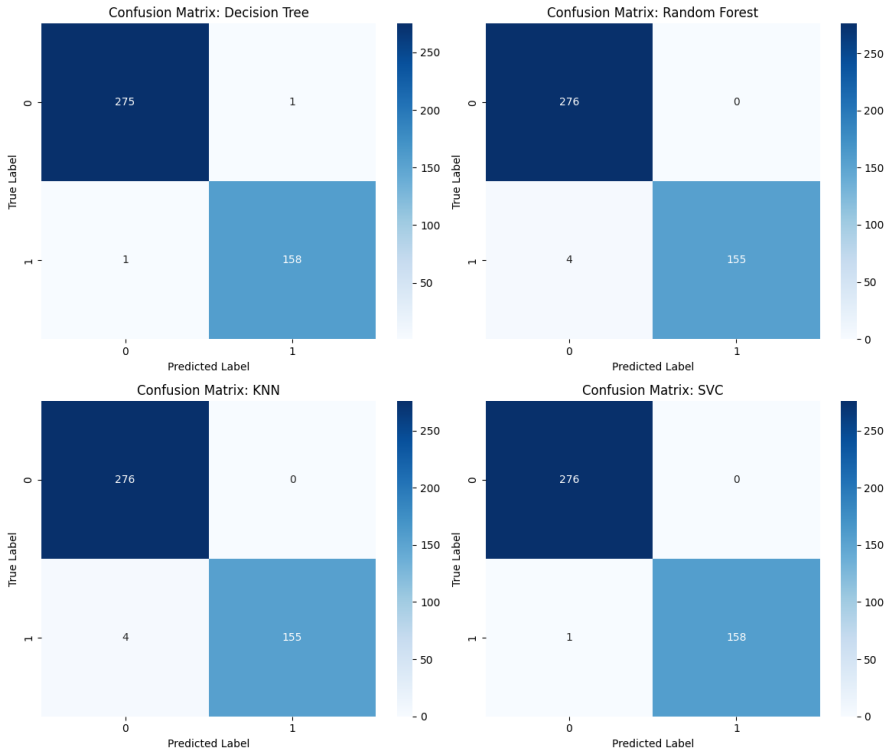

Figura 6 - Matrice di confusione - 15m attack - aggregation 5s - i=3 - undersampler

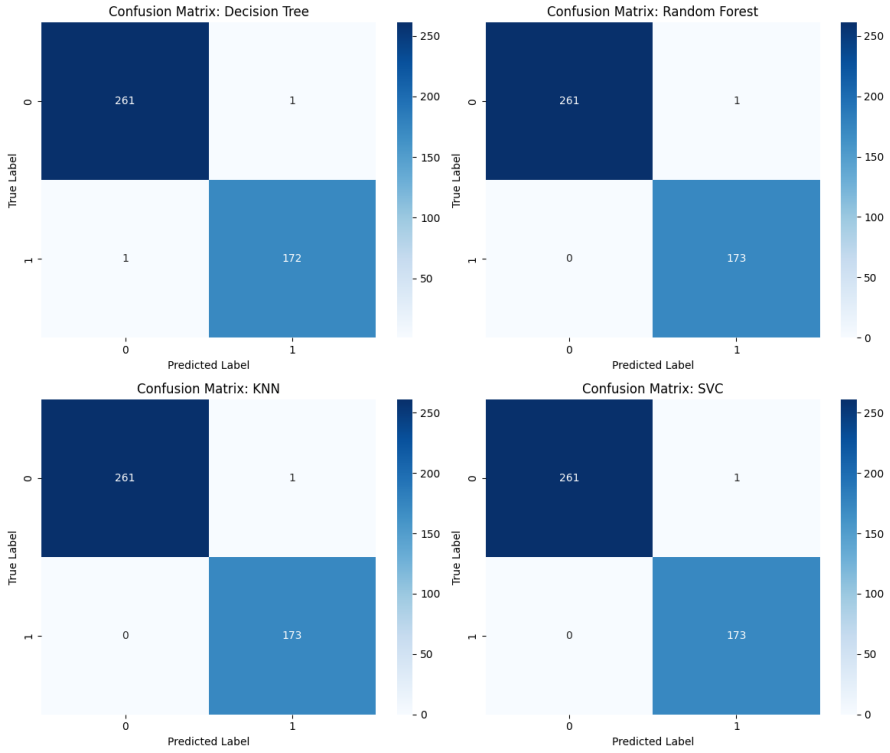

Figura 7 - Matrice di confusione - 15m attack - aggregation 5s - i=4 - undersampler

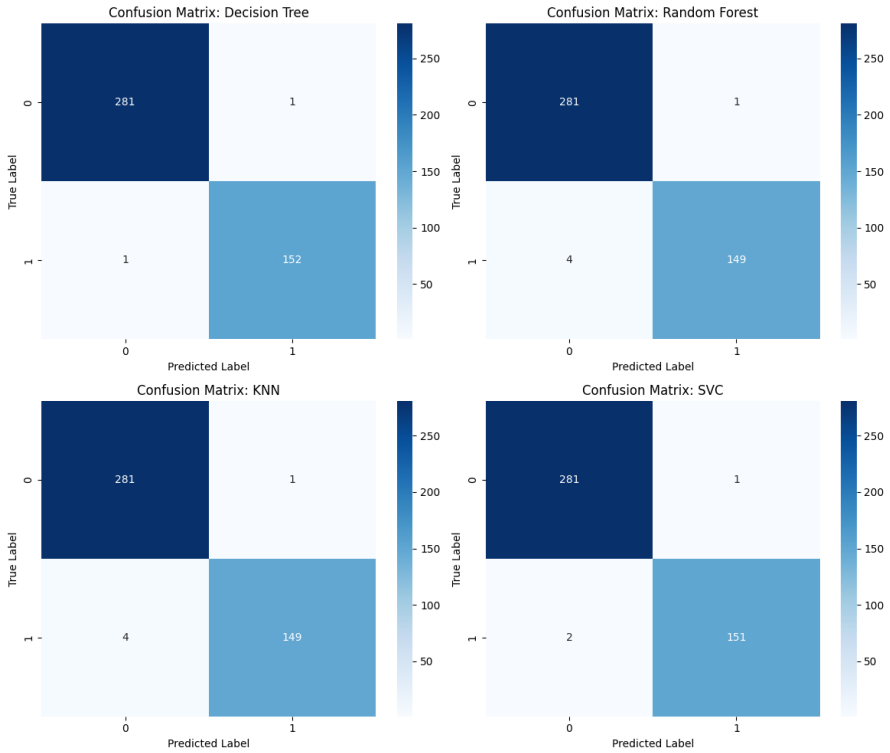

Figura 8 - Matrice di confusione - 15m attack - aggregation 5s - i=5 - undersampler

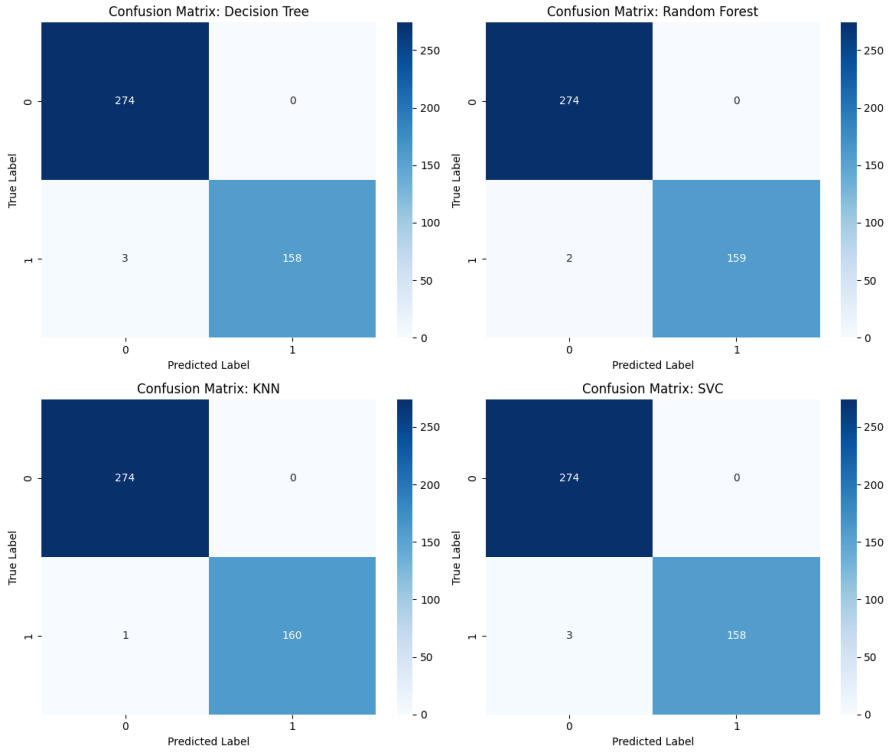

Figura 9 - Matrice di confusione - 15m attack - aggregation 5s - i=6 – undersampler

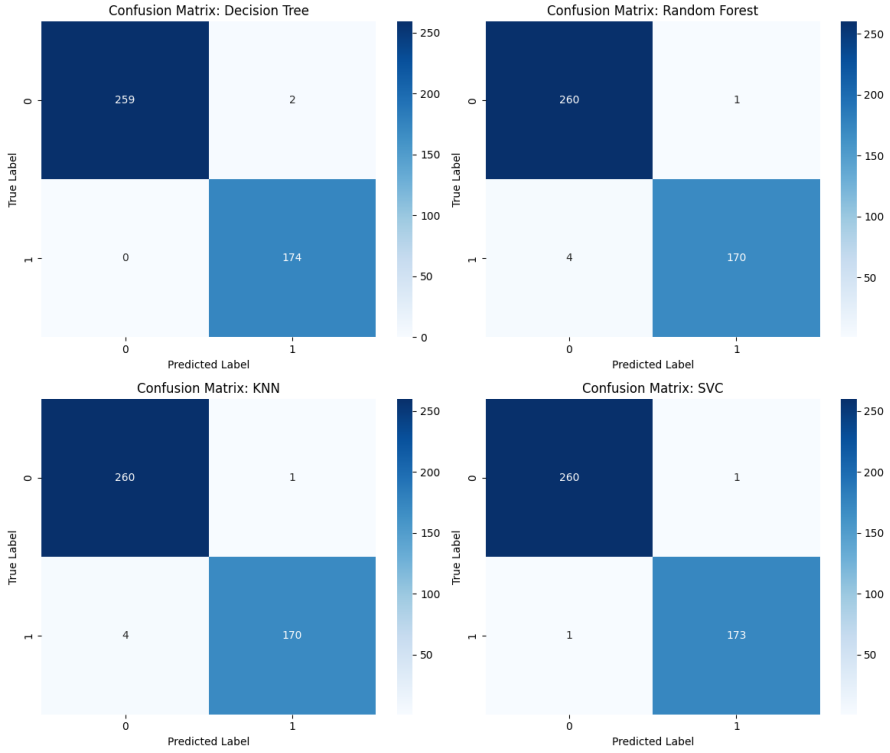

Figura 10 - Matrice di confusione - 15m attack - aggregation 5s - i=7 - undersampler

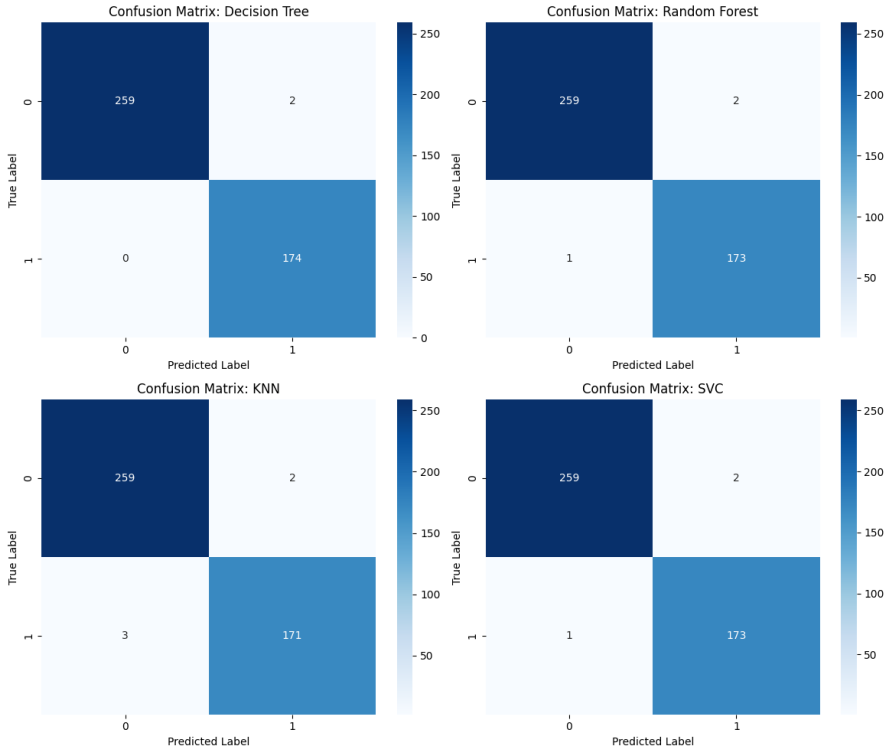

Figura 11 - Matrice di confusione - 15m attack - aggregation 5s - i=8 - undersampler

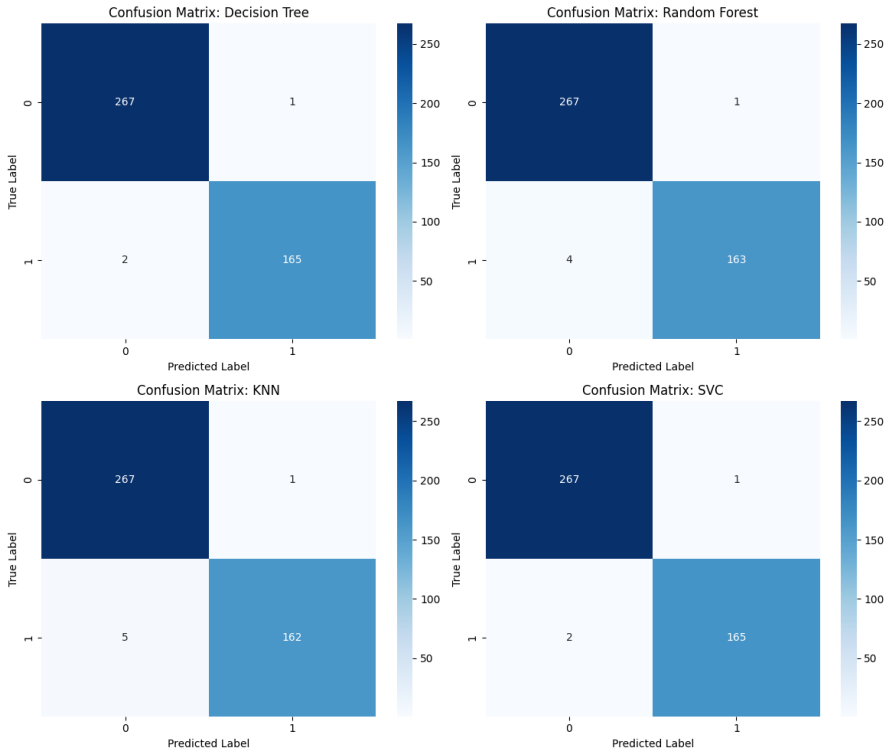

Figura 12 - Matrice di confusione - 15m attack - aggregation 5s - i=9 - undersampler

Dalle immagini di cui sopra si evince come il classificatore sia molto accurato, sbagliando nella classificazione di al massimo cinque campioni.

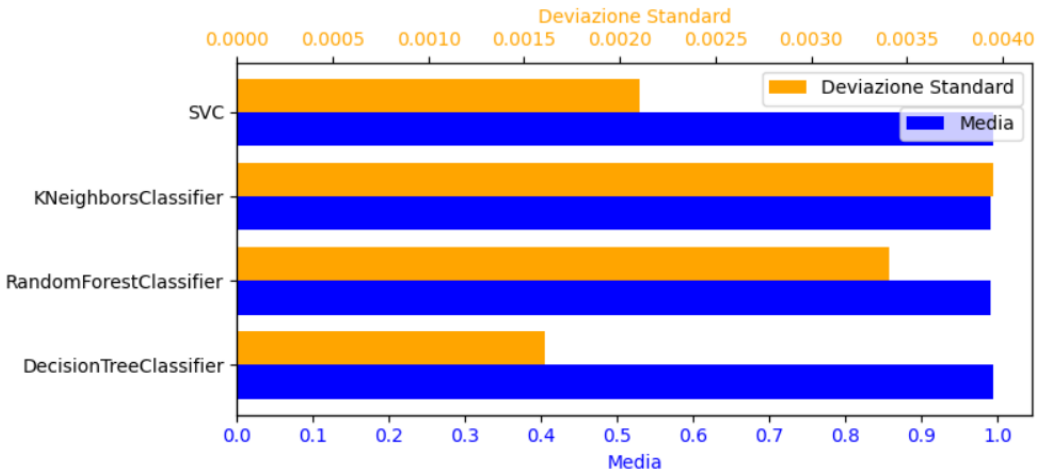

Figura 13 - Istogramma orizzontale - Media e deviazione standard accuracy - undersampler

La media delle accuracy è molto vicina a 1 in quanto le accuracy dei modelli nelle singole iterazioni sono molto elevate; infatti, esse assumono valori nell'intorno di 0.99.
Le deviazioni standard delle accuracy sono molto basse, il che significa che nelle diverse iterazioni esse non variano di molto.
I risultati, dunque, sono più o meno costanti tra le varie iterazioni.
In particolare, il modello che risulta più stabile in termini di accuratezza è il Decision Tree, mentre quello che presenta risultati più variabili è il KNN.

CATTURA PING FLOODING 15M ATTACK - AGGREGATION 5S - SENZA UNDERSAMPLER

In questo caso ai modelli viene sottoposto un dataset sbilanciato, per poi confrontare i risultati ottenuti con il caso precedente per analizzare quanto lo sbilanciamento di classe li influenza.
Le matrici di confusione nelle dieci iterazioni del caso in questione verranno illustrate nel seguito.

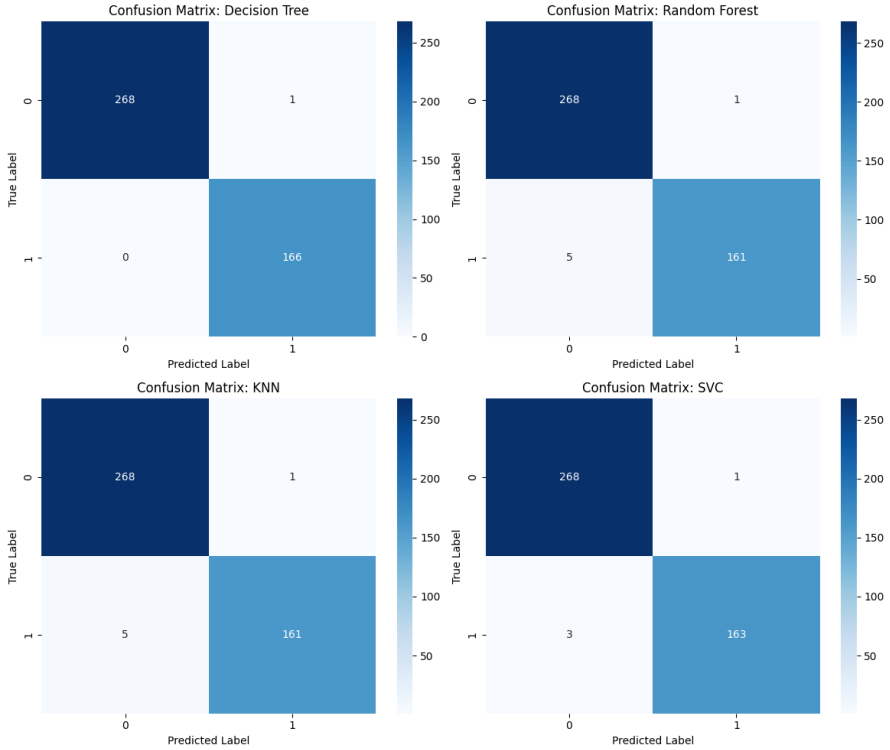

Figura 14 - Matrice di confusione - 15m attack - aggregation 5s - i=0 - no undersampler

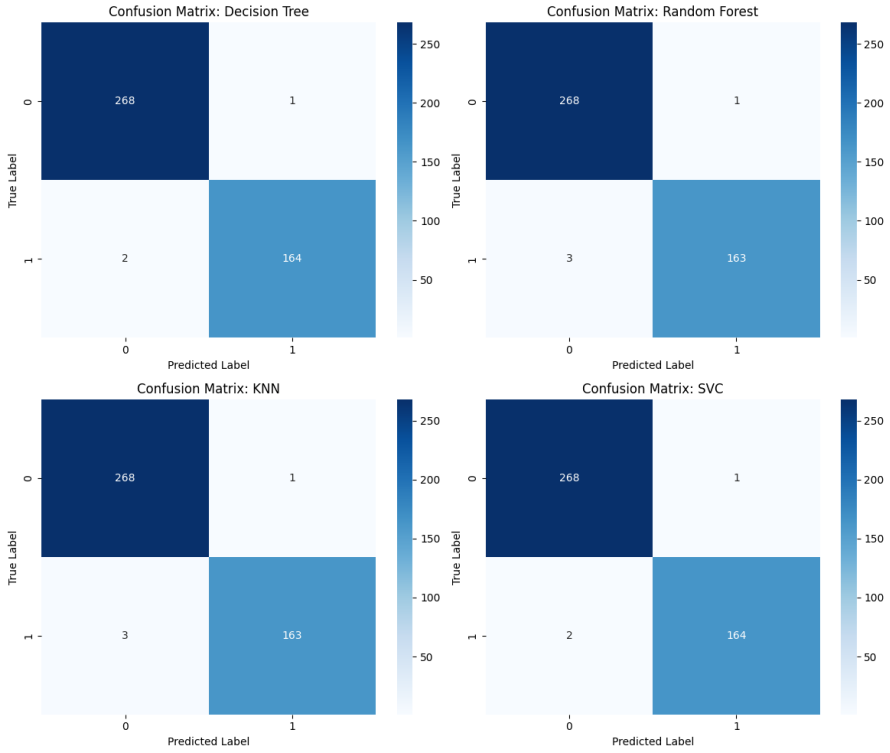

Figura 15 - Matrice di confusione - 15m attack - aggregation 5s - i=1 - no undersampler

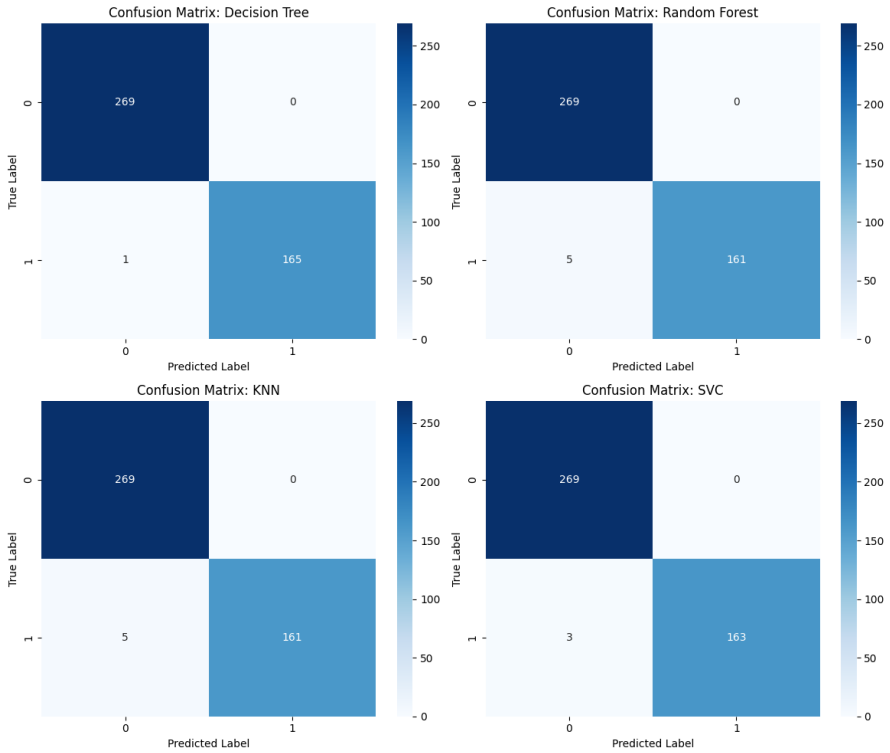

Figura 16 - Matrice di confusione - 15m attack - aggregation 5s - i=2 - no undersampler

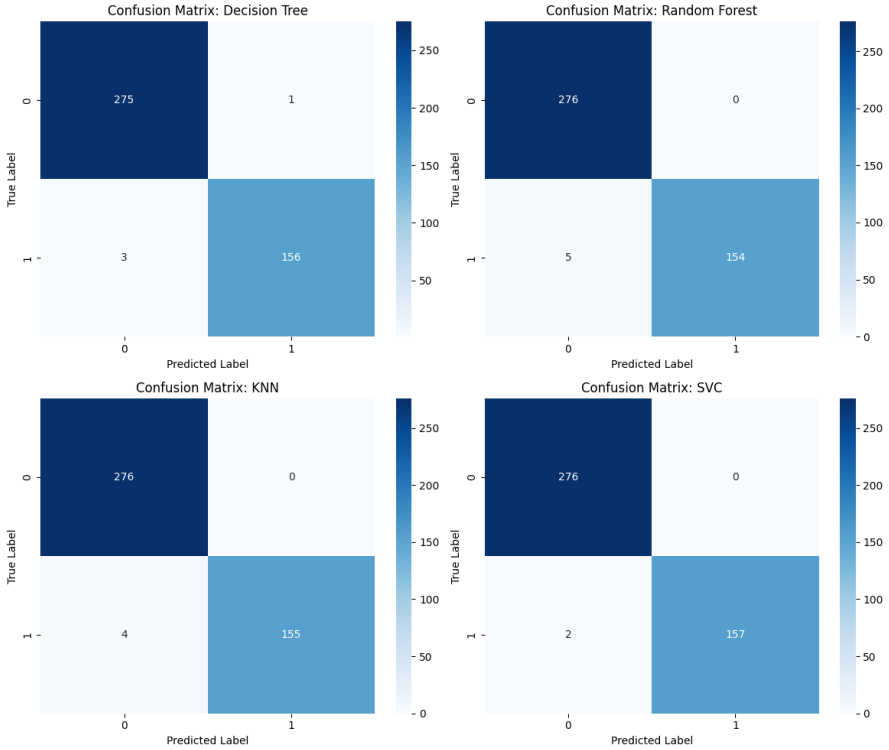

Figura 17 - Matrice di confusione - 15m attack - aggregation 5s - i=3 - no undersampler

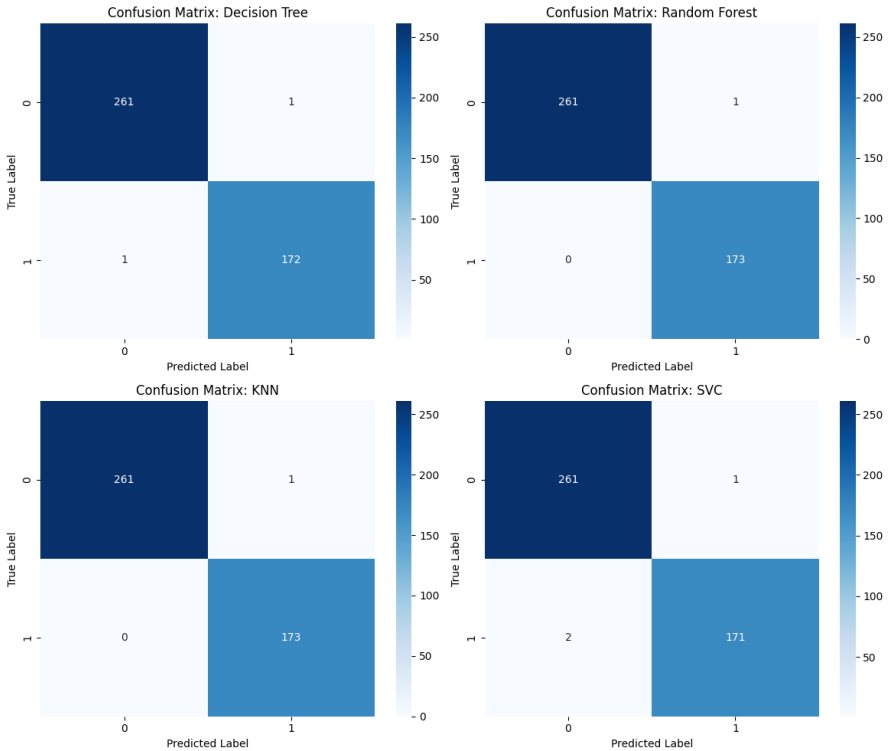

Figura 18 - Matrice di confusione - 15m attack - aggregation 5s - i=4 - no undersampler

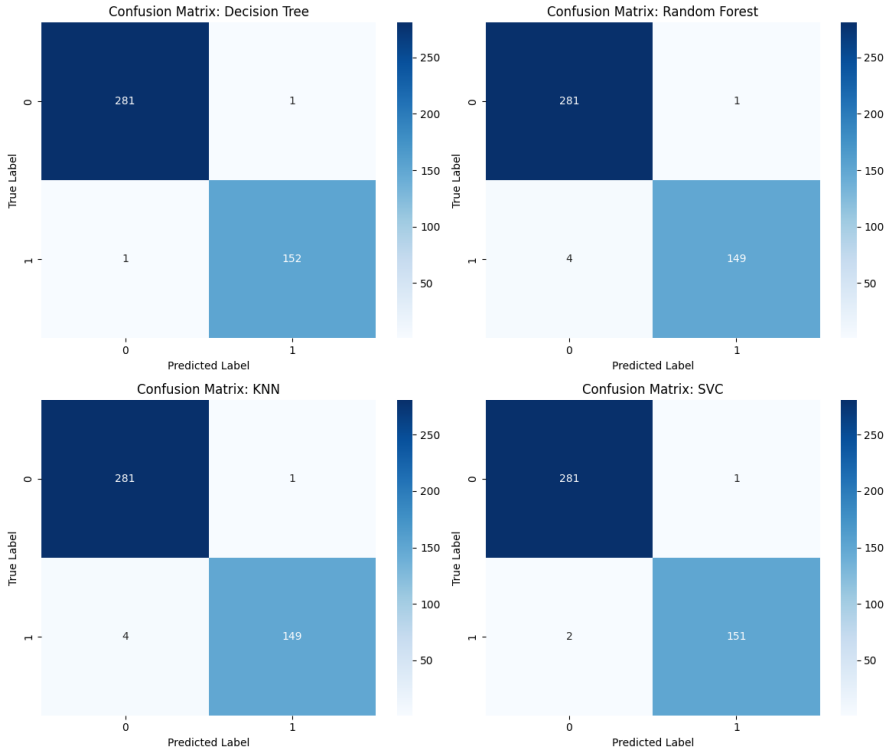

Figura 19 - Matrice di confusione - 15m attack - aggregation 5s - i=5 - no undersampler

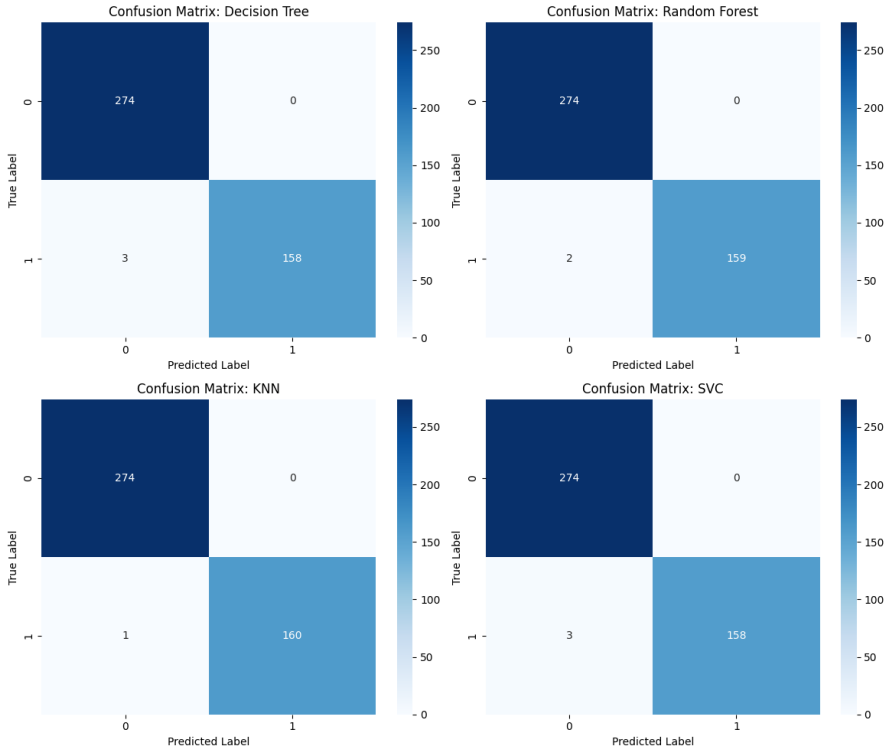

Figura 20 - Matrice di confusione - 15m attack - aggregation 5s - i=6 - no undersampler

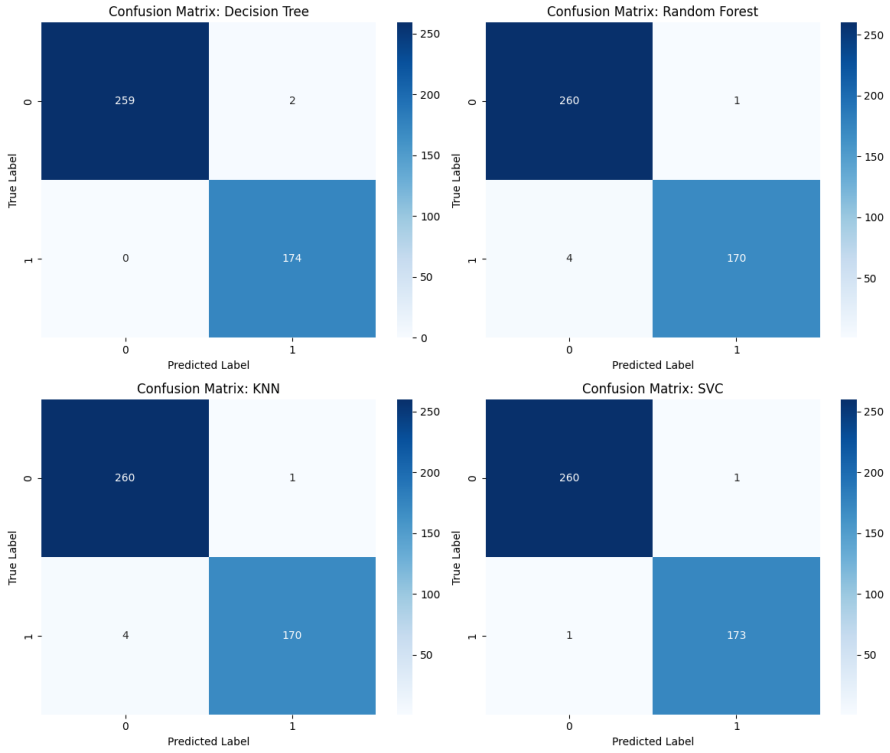

Figura 21 - Matrice di confusione - 15m attack - aggregation 5s - i=7 - no undersampler

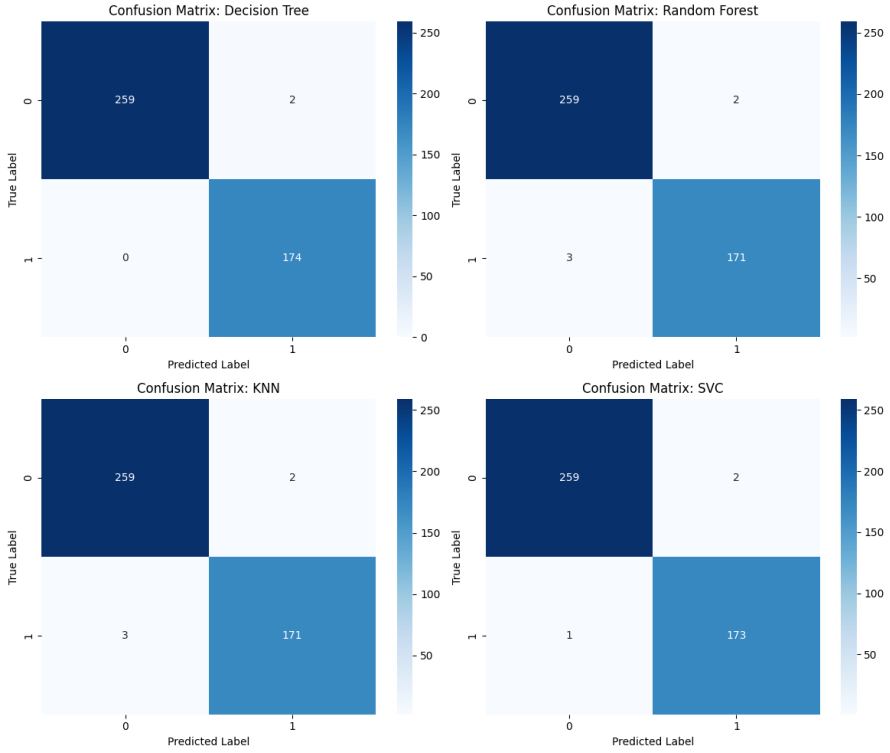

Figura 22 - Matrice di confusione - 15m attack - aggregation 5s - i=8 - no undersampler

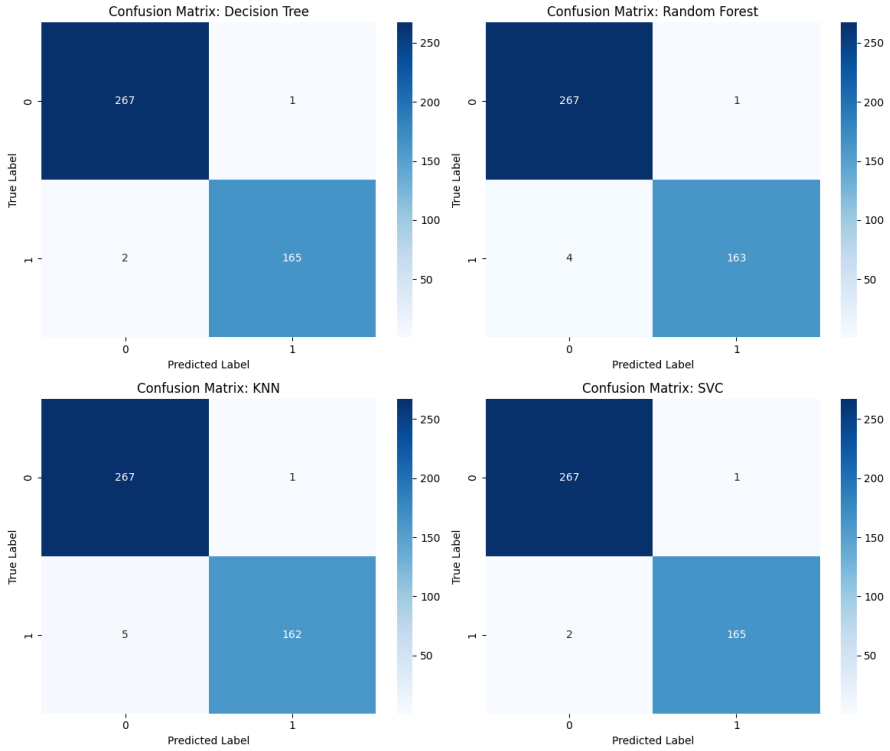

Figura 23 - Matrice di confusione - 15m attack - aggregation 5s - i=9 - no undersampler

Dalle matrici di confusione si nota come l'introduzione dell'undersampler, nel caso precedente, non abbia migliorato o peggiorato le prestazioni dei modelli. Essi riescono a classificare bene indipendentemente dallo sbilanciamento del dataset.

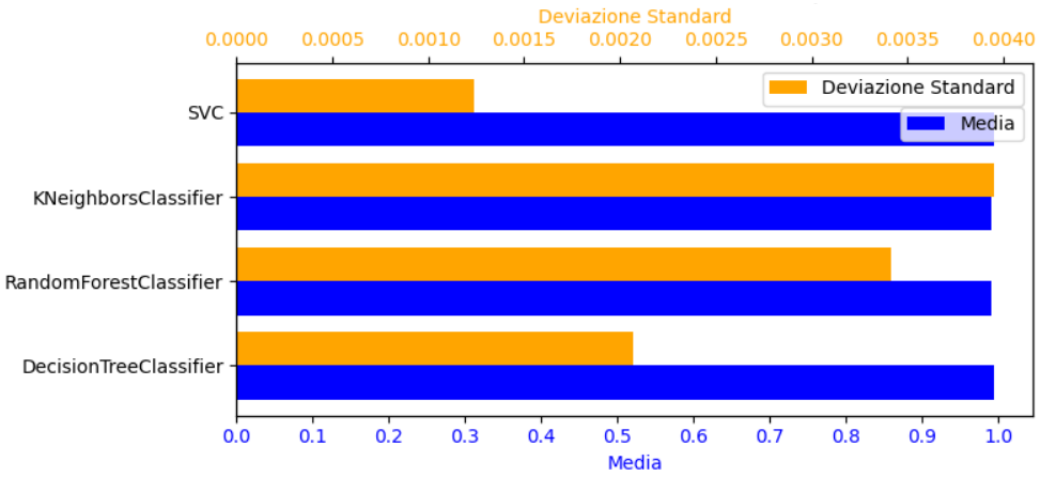

Figura 24 - Istogramma orizzontale - Media e deviazione standard accuracy - no undersampler

Il modello KNN continua a essere il modello che mostra una maggiore variabilità dell'accuracy ottenute nelle singole iterazioni.
Questa volta il modello SVC è il modello che mostra una minore variabilità dell'accuratezza.
I modelli, complessivamente, mostrano una media delle accuracy nelle dieci iterazioni che si avvicina a uno, come nel caso precedente.

CATTURA PING FLOODING 5M ATTACK - AGGREGATION 5S - CON UNDERSAMPLER

In questo caso viene sottoposto ai modelli un dataset bilanciato che contiene attacchi ping flooding che si protraggono per 5m.
Si vuole analizzare i risultati dei modelli, considerando di fornire un set di dati contenente un numero minore di campioni della classe di catture malevole.

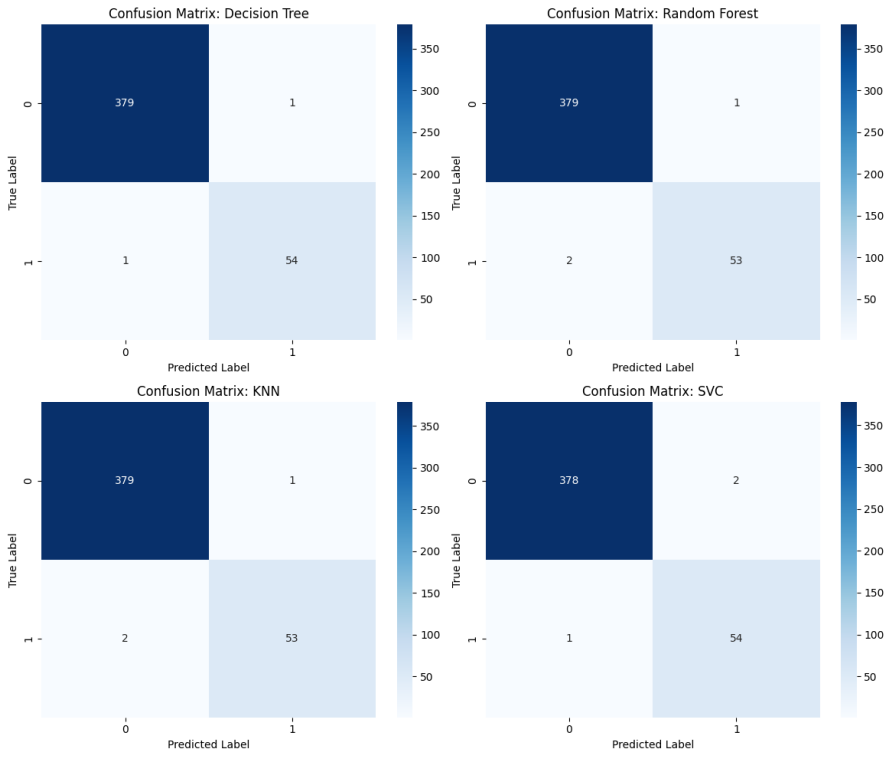

Figura 25 - Matrice di confusione - 5m attack - aggregation 5s - i=0 – undersampler

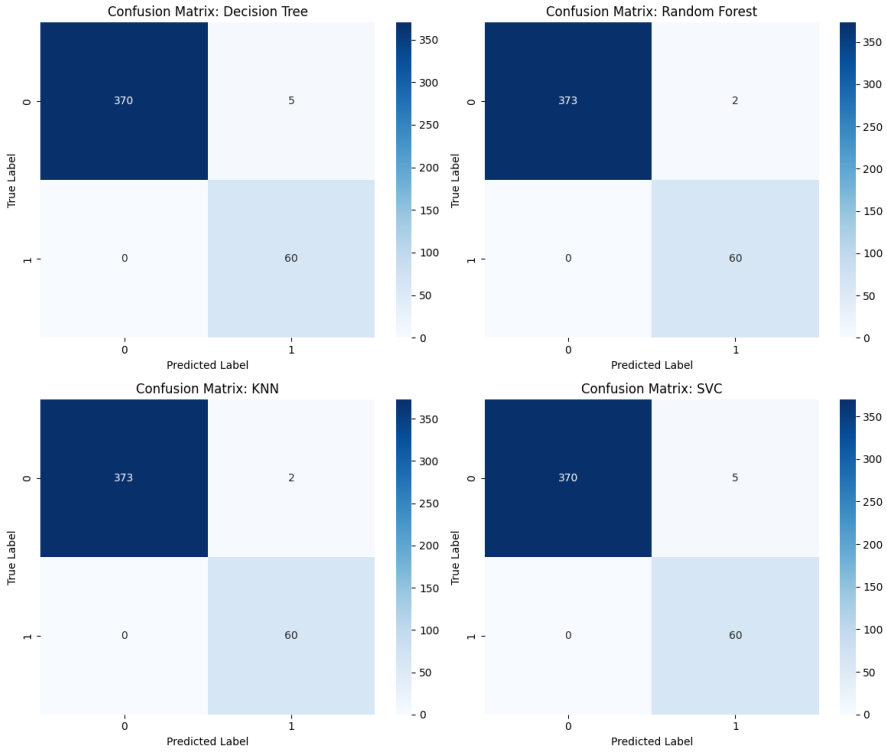

Figura 26 - Matrice di confusione - 5m attack - aggregation 5s - i=1 – undersampler

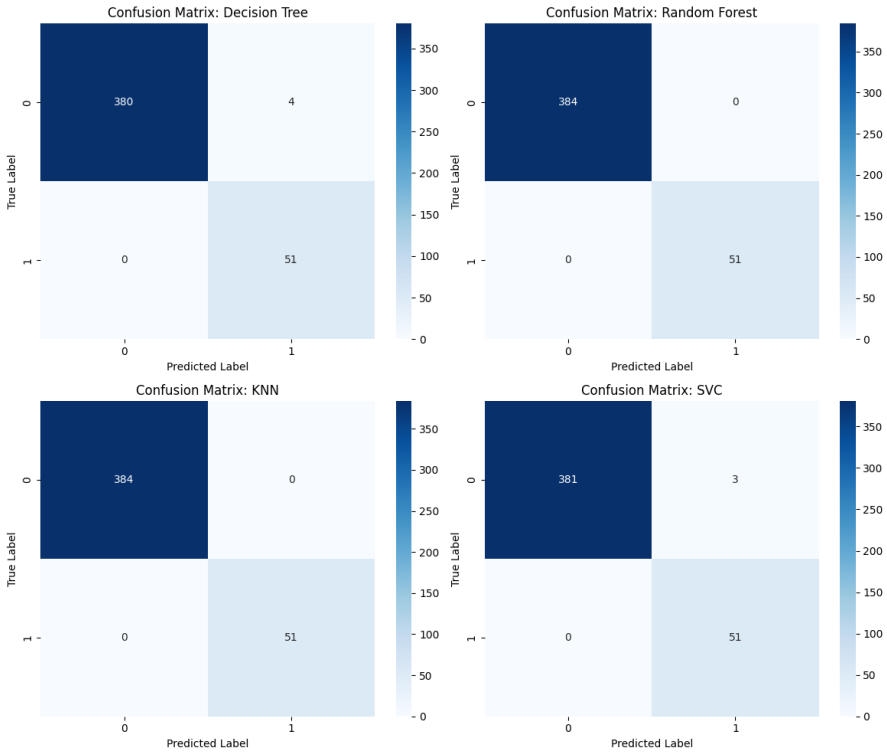

Figura 27 - Matrice di confusione - 5m attack - aggregation 5s - i=2 – undersampler

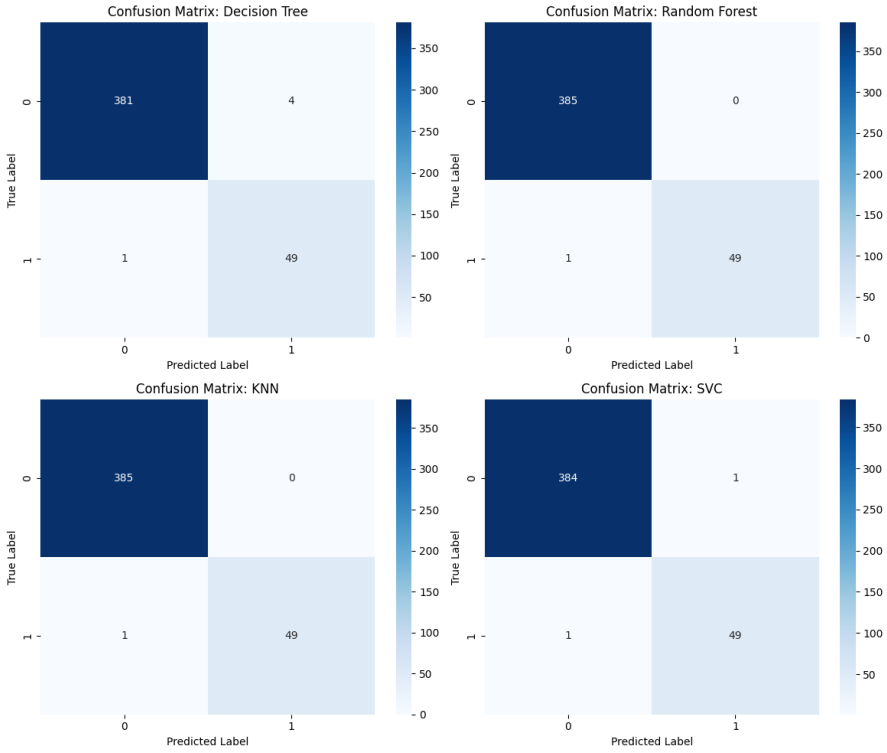

Figura 28 - Matrice di confusione - 5m attack - aggregation 5s - i=3 – undersampler

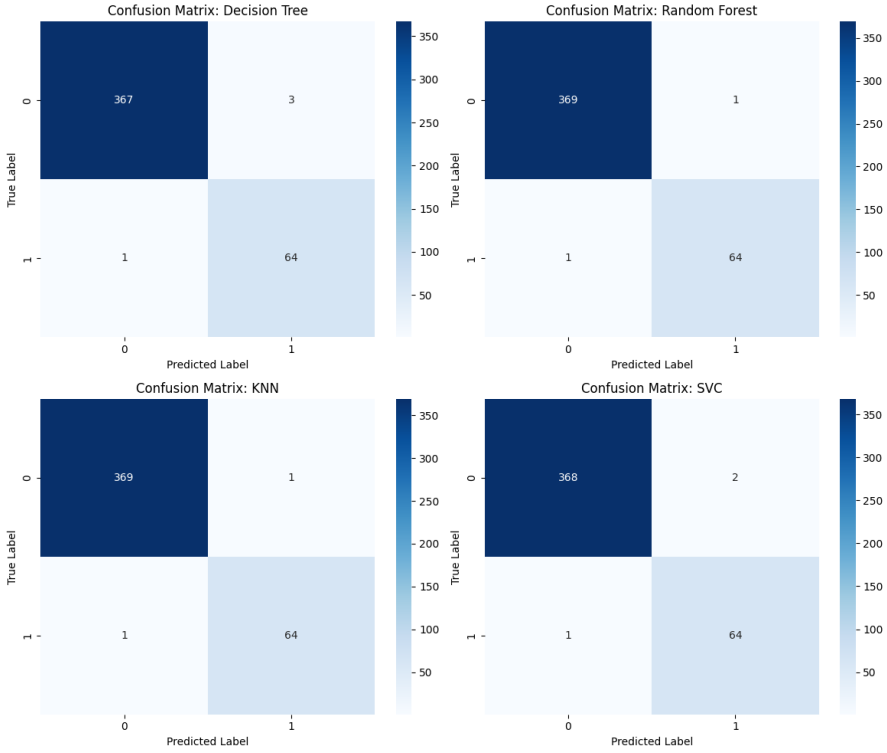

Figura 29 - Matrice di confusione - 5m attack - aggregation 5s - i=4 – undersampler

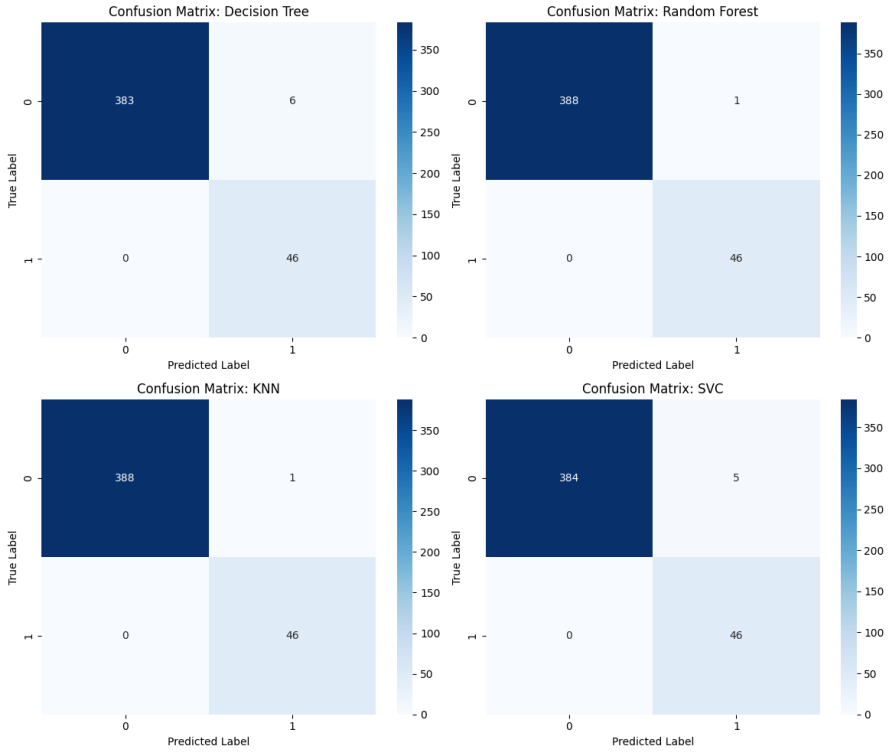

Figura 30 - Matrice di confusione - 5m attack - aggregation 5s - i=5 – undersampler

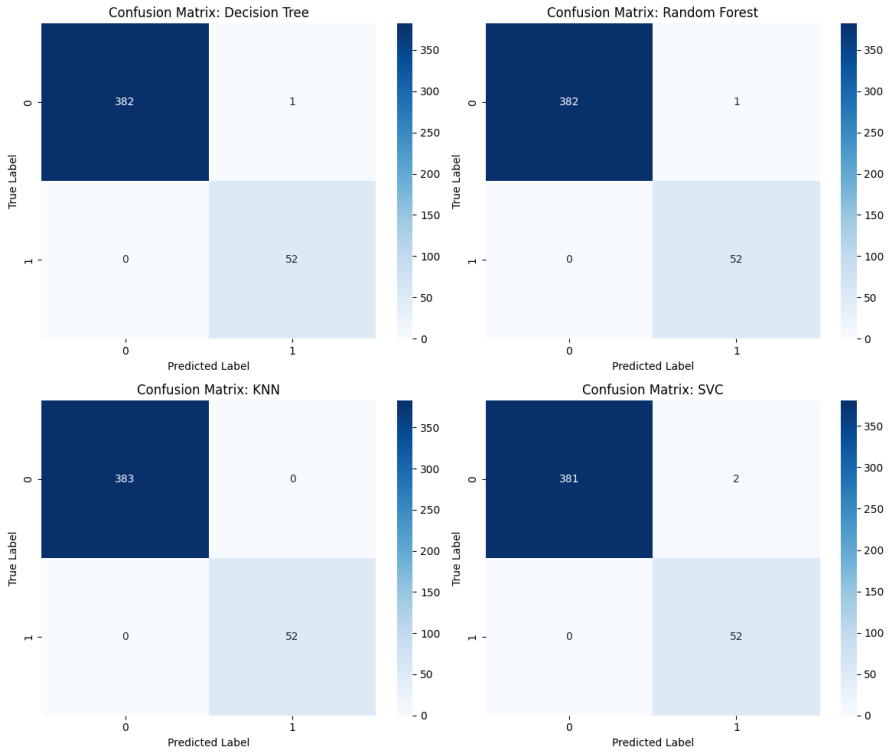

Figura 31 - Matrice di confusione - 5m attack - aggregation 5s - i=6 – undersampler

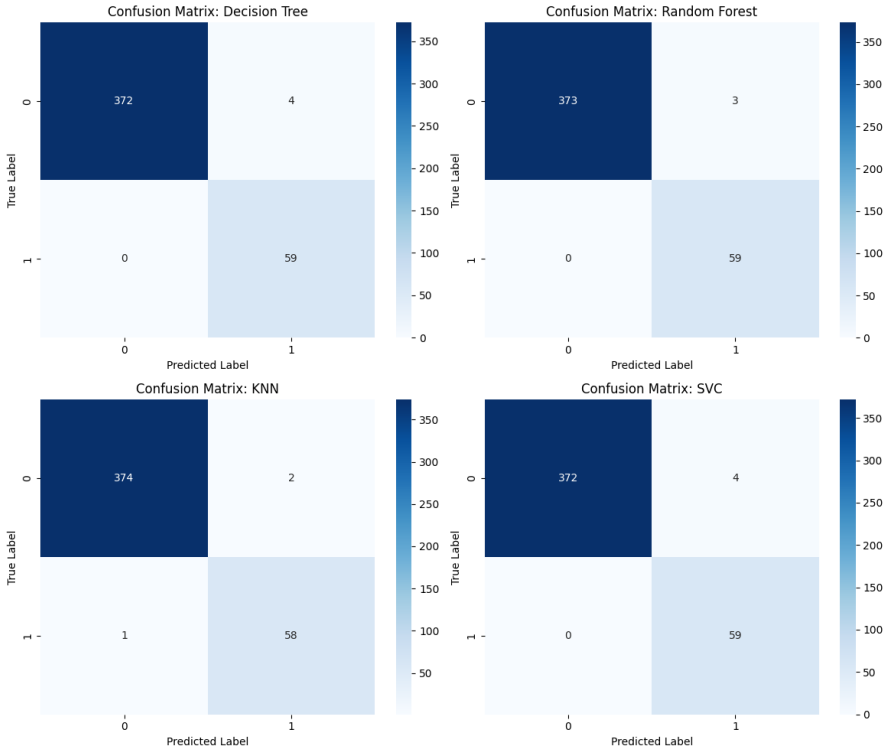

Figura 32 - Matrice di confusione - 5m attack - aggregation 5s - i=7 – undersampler

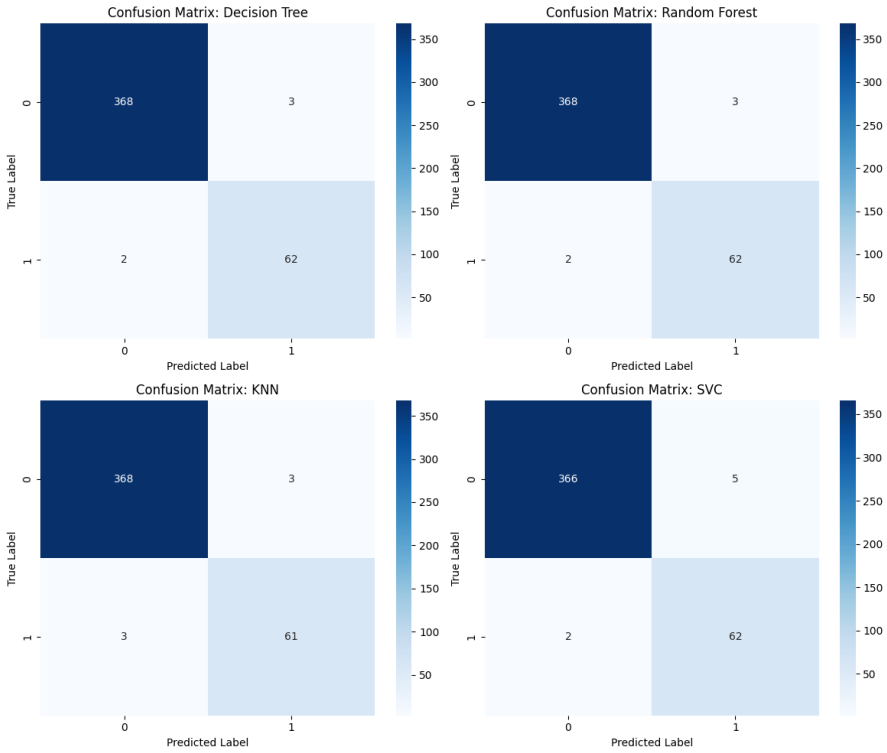

Figura 33 - Matrice di confusione - 5m attack - aggregation 5s - i=8 – undersampler

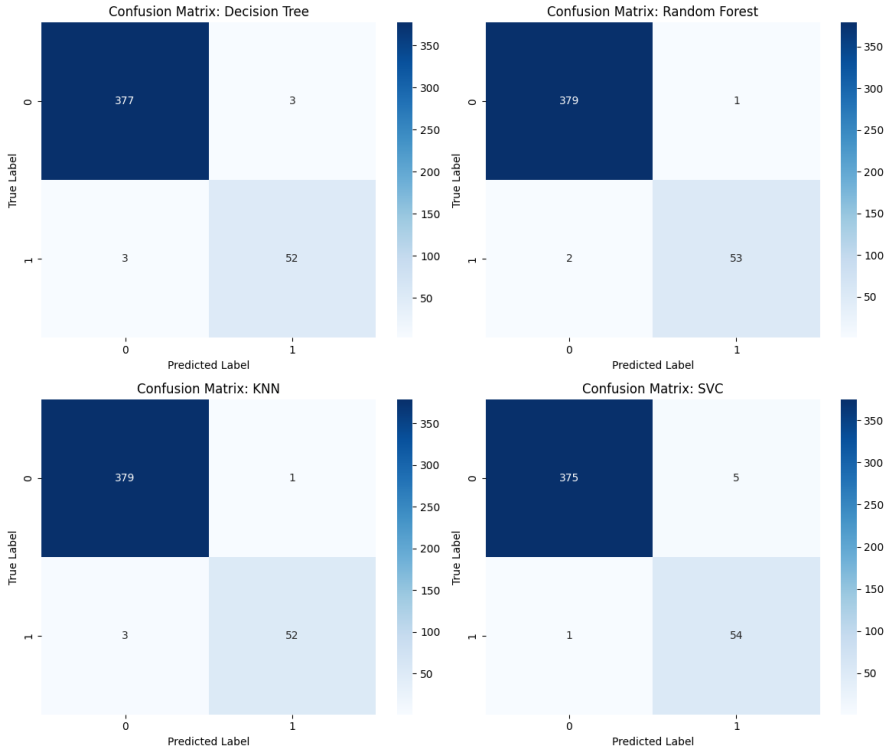

Figura 34 - Matrice di confusione - 5m attack - aggregation 5s - i=9 – undersampler

Diminuendo il numero di campioni della classe minoritaria (classe delle catture malevole) ci si aspetterebbe un peggioramento delle prestazioni dei diversi modelli ma i risultati smentiscono le aspettative. I modelli riescono ancora una volta a classificare correttamente, talvolta in maniera perfetta i campioni, ossia le catture pulite e le quelle malevole.

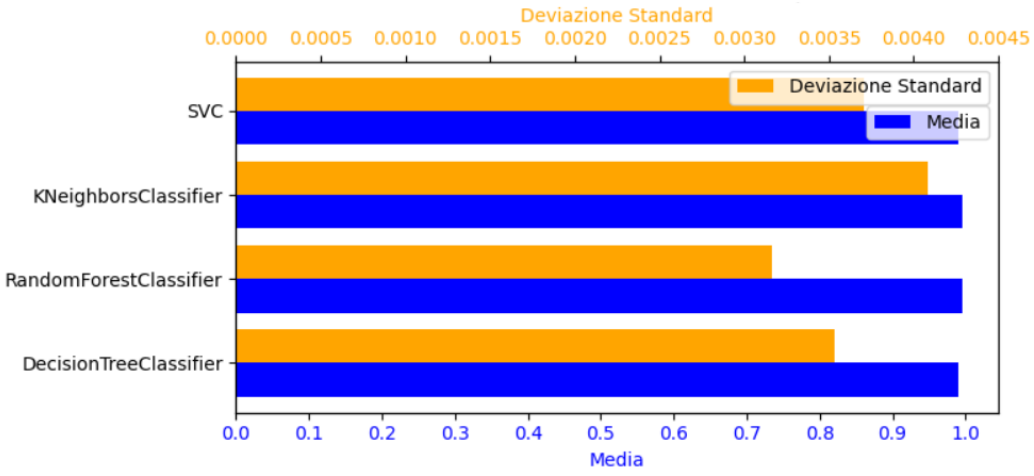

Figura 35 - Istogramma orizzontale - Media e deviazione standard accuracy – undersampler

Ciò è testimoniato da medie che tendono a uno e da deviazioni standard nell'ordine dei millesimi.

CATTURA PING FLOODING 5M ATTACK - AGGREGATION 5S - SENZA UNDERSAMPLER

Di seguito vengono illustrate le matrici di confusione nel caso dell'attacco ping flooding con durata di 5m e senza l'utilizzo dell'undersampler.

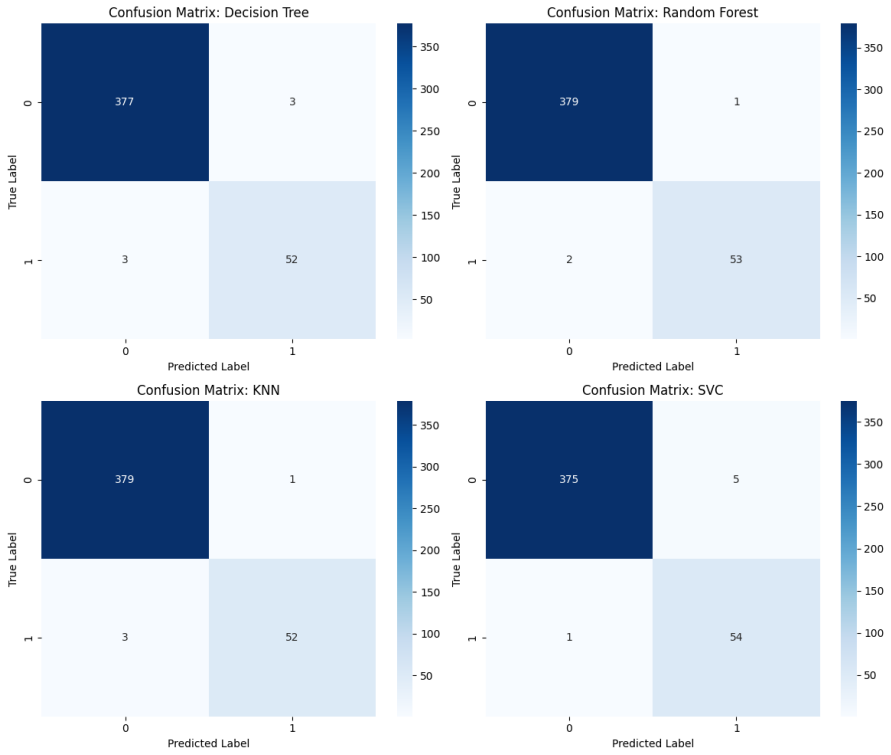

Figura 36 - Matrice di confusione - 5m attack - aggregation 5s - i=0 - no undersampler

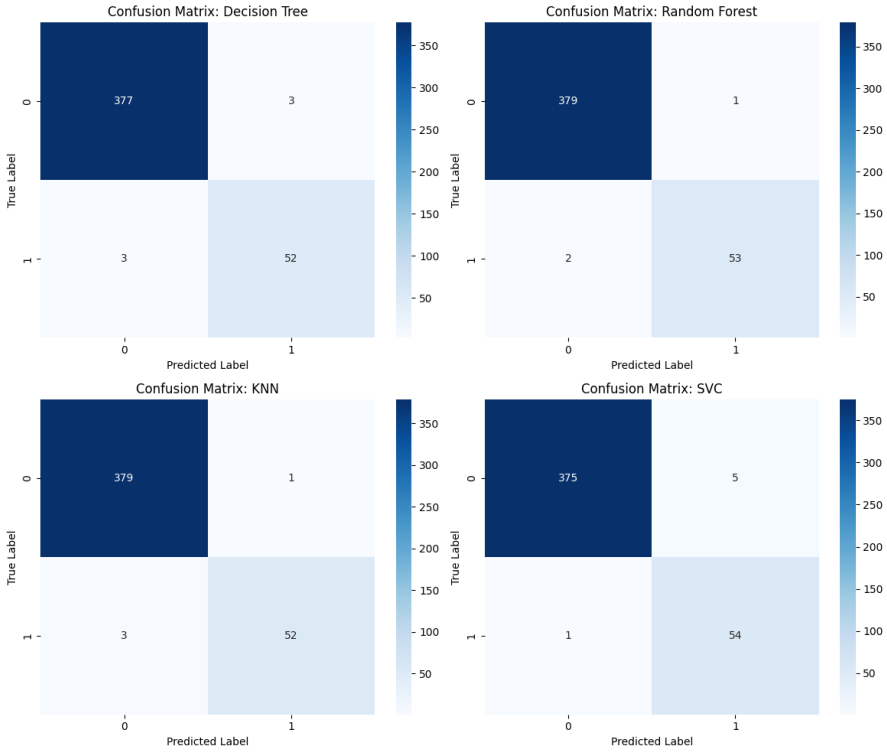

Figura 37 - Matrice di confusione - 5m attack - aggregation 5s - i=1 - no undersampler

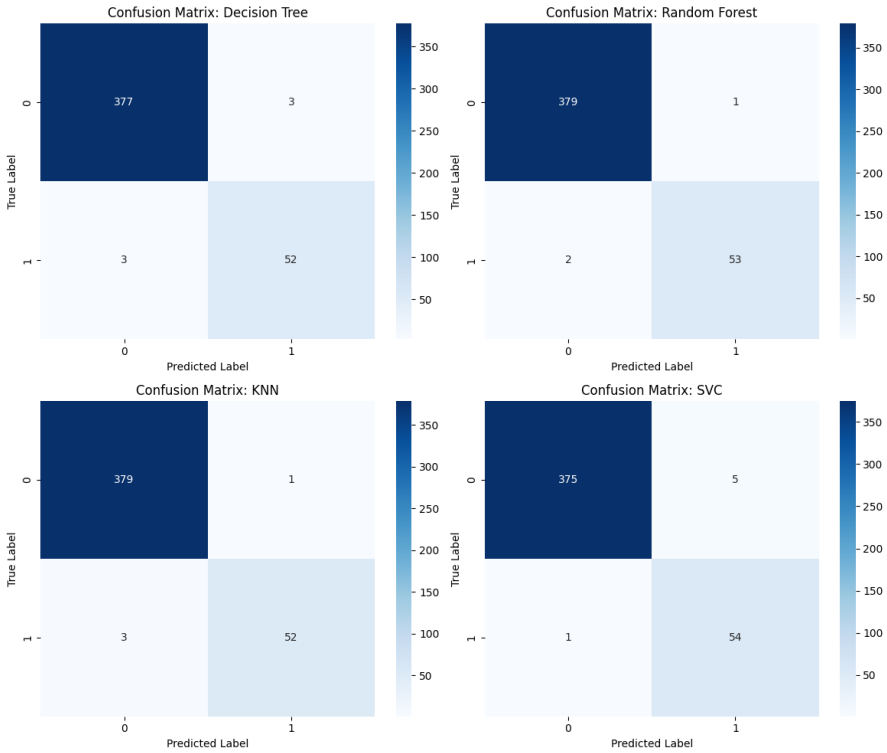

Figura 38 - Matrice di confusione - 5m attack - aggregation 5s - i=2 - no undersampler

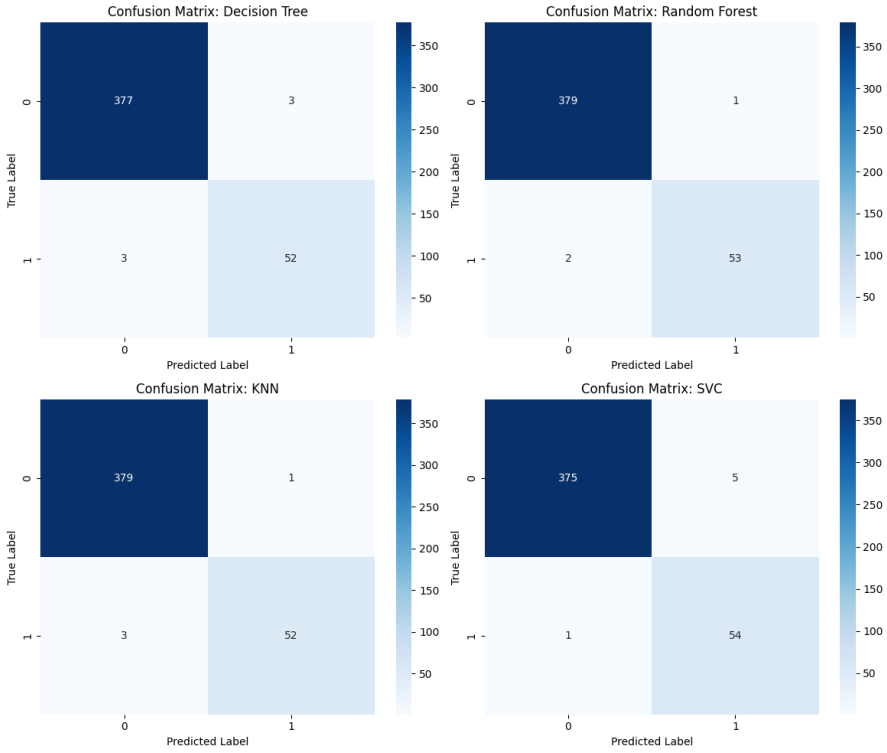

Figura 39 - Matrice di confusione - 5m attack - aggregation 5s - i=3 - no undersampler

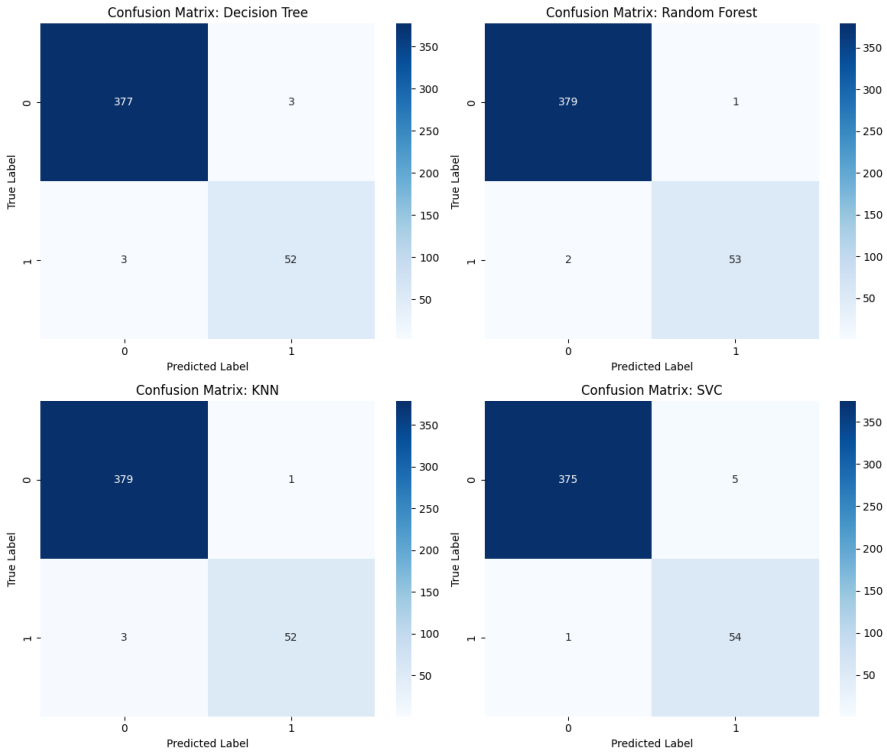

Figura 40 - Matrice di confusione - 5m attack - aggregation 5s - i=4 - no undersampler

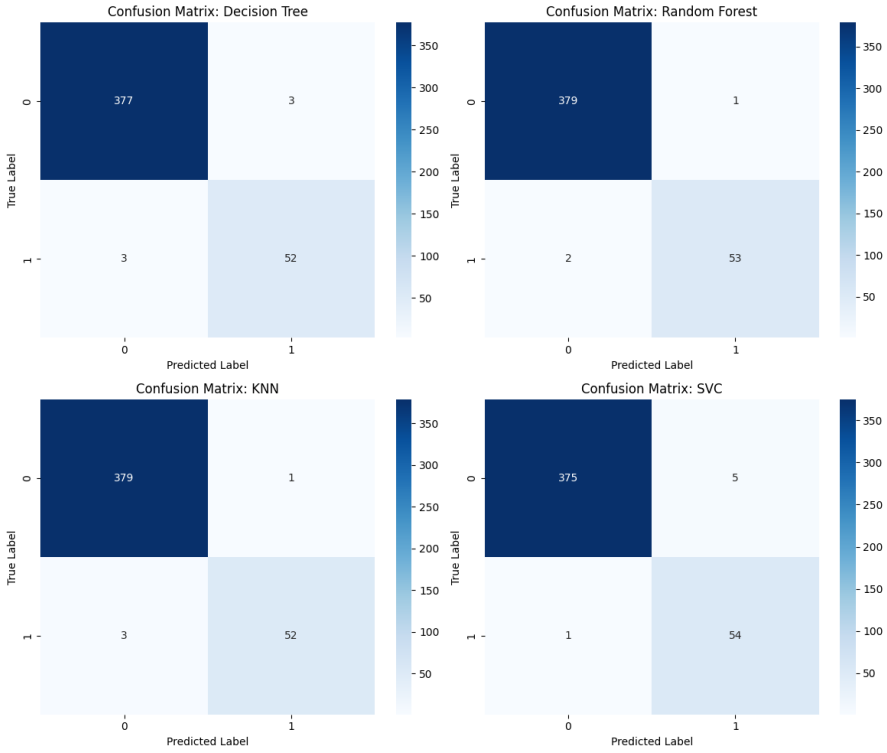

Figura 41 - Matrice di confusione - 5m attack - aggregation 5s - i=5 - no undersampler

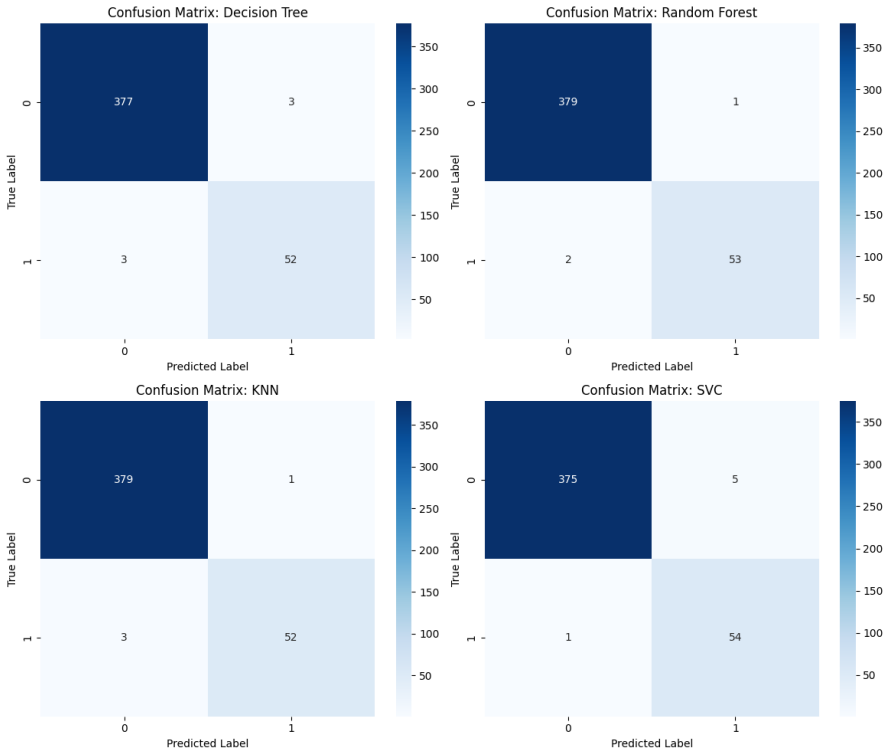

Figura 42 - Matrice di confusione - 5m attack - aggregation 5s - i=6 - no undersampler

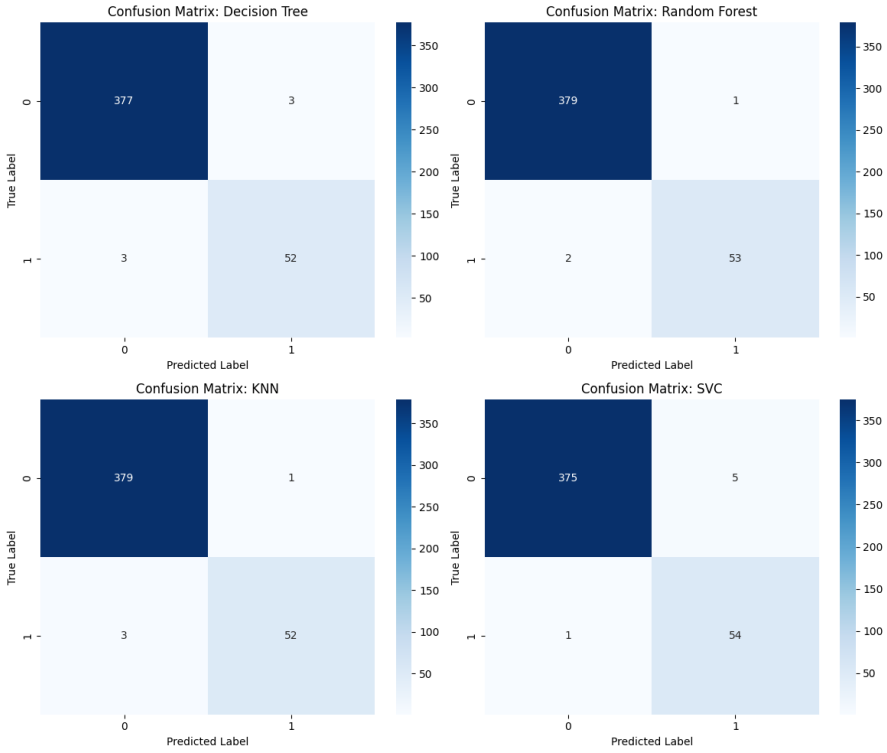

Figura 43 - Matrice di confusione - 5m attack - aggregation 5s - i=7 - no undersampler

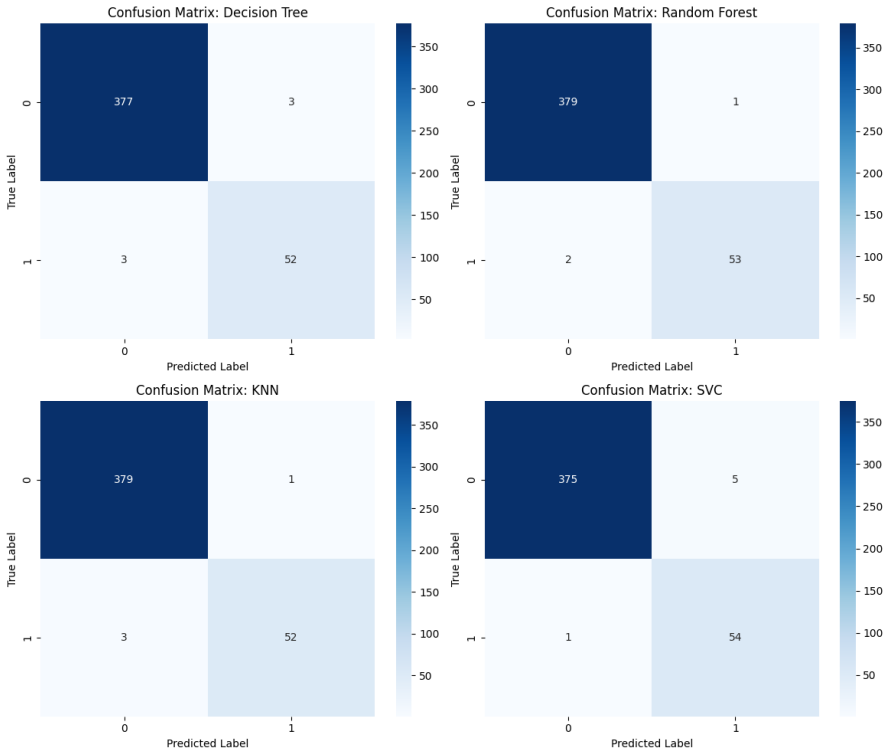

Figura 44 - Matrice di confusione - 5m attack - aggregation 5s - i=8 - no undersampler

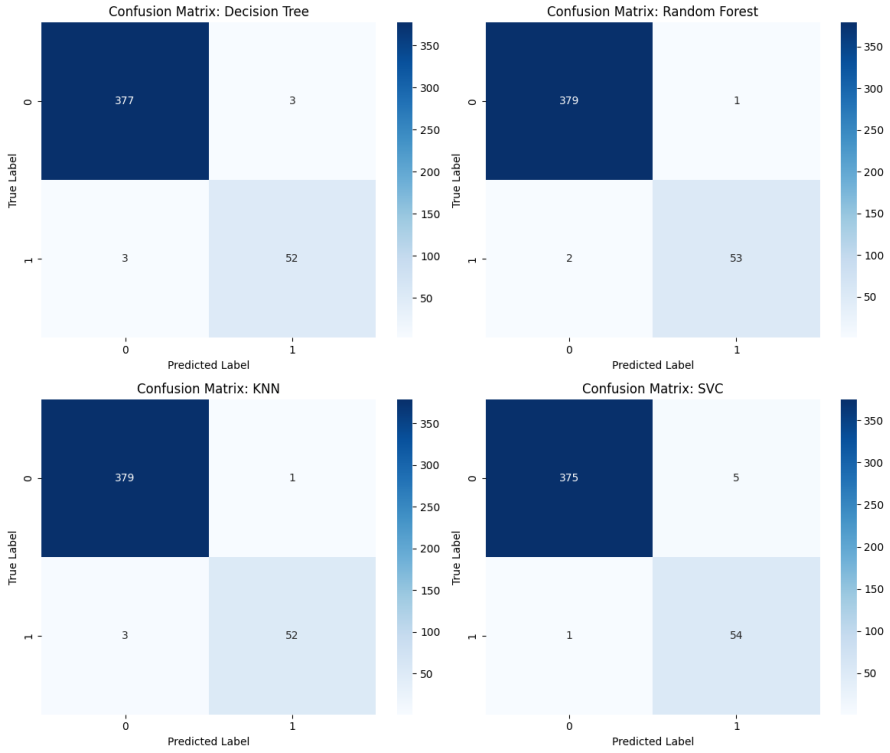

Figura 45 - Matrice di confusione - 5m attack - aggregation 5s - i=9 - no undersampler

Le prestazioni dei modelli nel caso corrente sono in linea con il caso del dataset bilanciato.

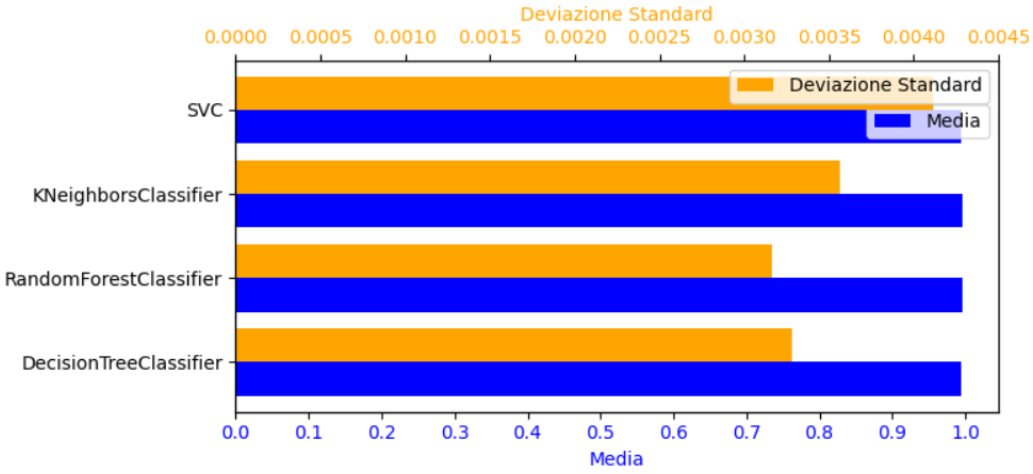

Figura 46 - Istogramma orizzontale - Media e deviazione standard accuracy - no undersampler

Indipendentemente da quale sia il modello che mostra una maggiore variabilità dell'accuracy, tutti presentano risultati più che soddisfacenti.

CATTURA PING FLOODING 1M ATTACK - AGGREGATION 5S - CON UNDERSAMPLER

Le seguenti matrici di confusione sono state ottenute fornendo in input ai modelli il dataset bilanciato dall'undersampler contenenti catture malevole dalla durata complessiva di 1m.

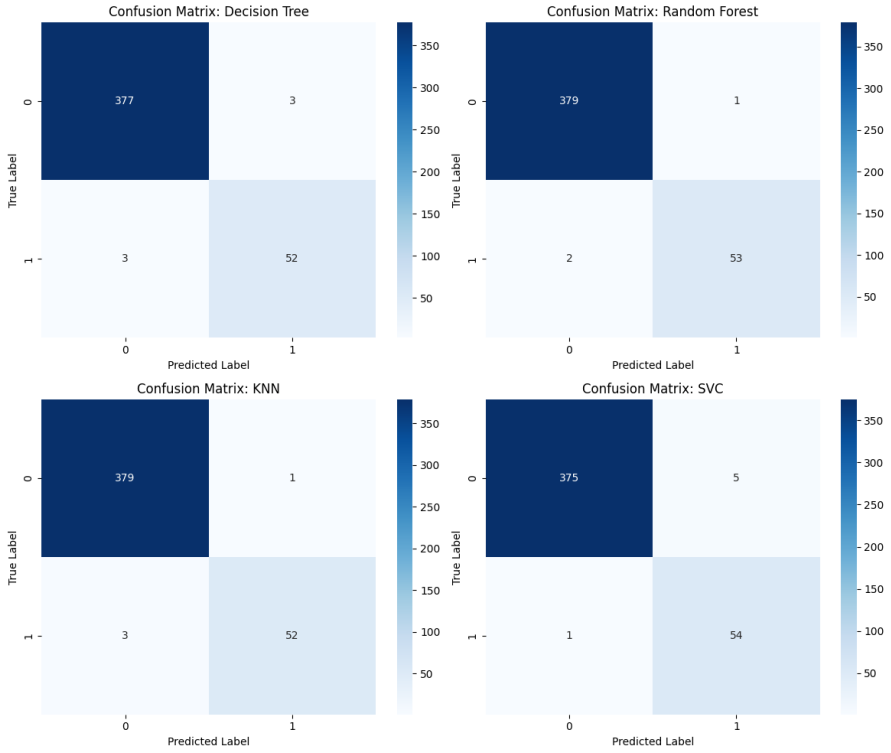

Figura 47 - Matrice di confusione - 1m attack - aggregation 5s - i=0 - no undersampler

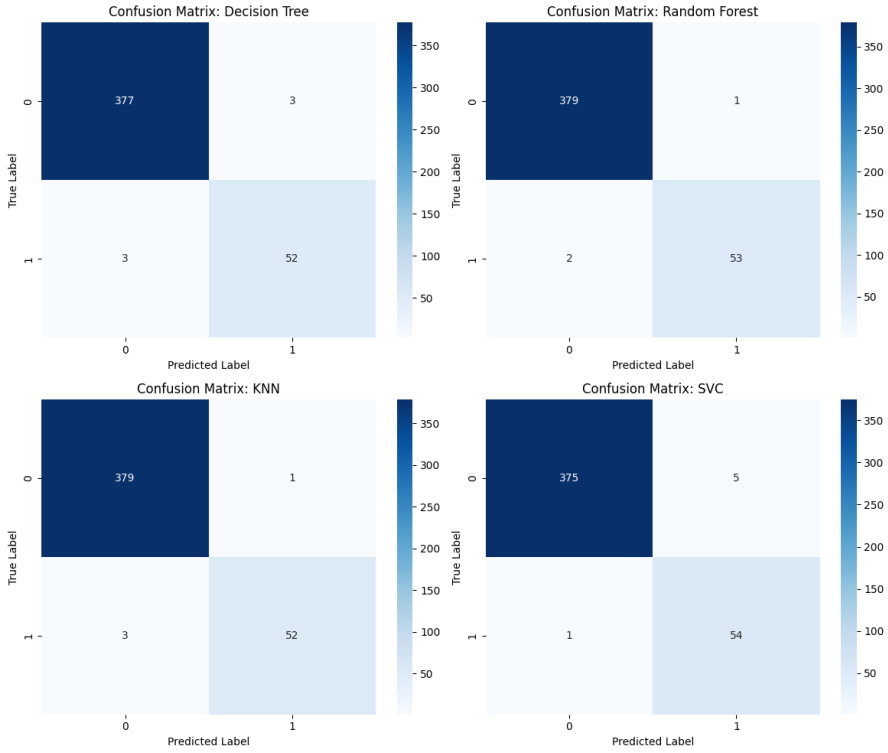

Figura 48 - Matrice di confusione - 1m attack - aggregation 5s - i=1 – undersampler

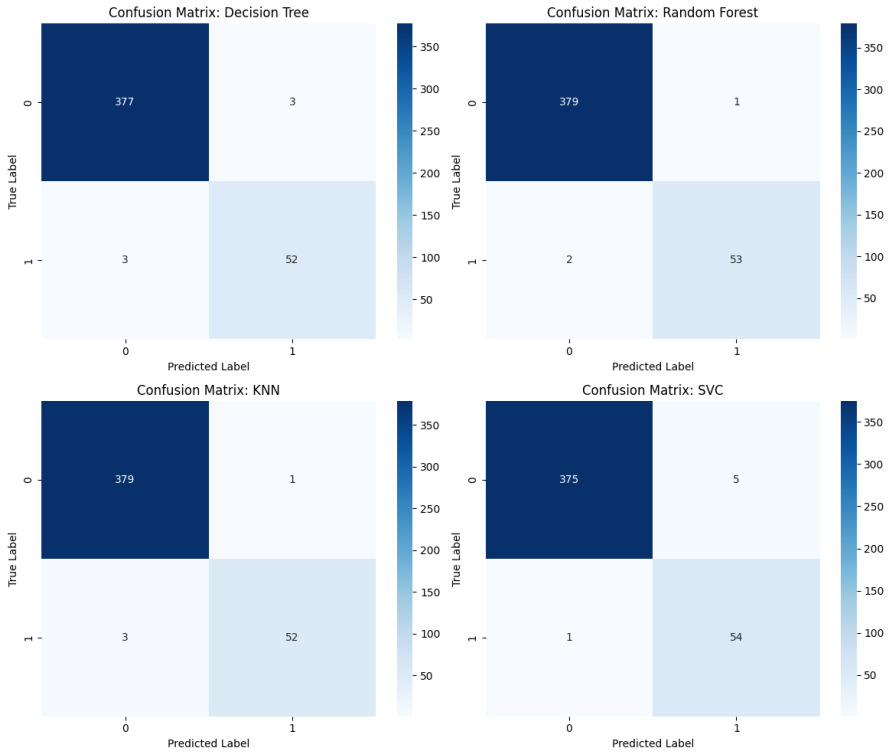

Figura 49 - Matrice di confusione - 1m attack - aggregation 5s - i=2 – undersampler

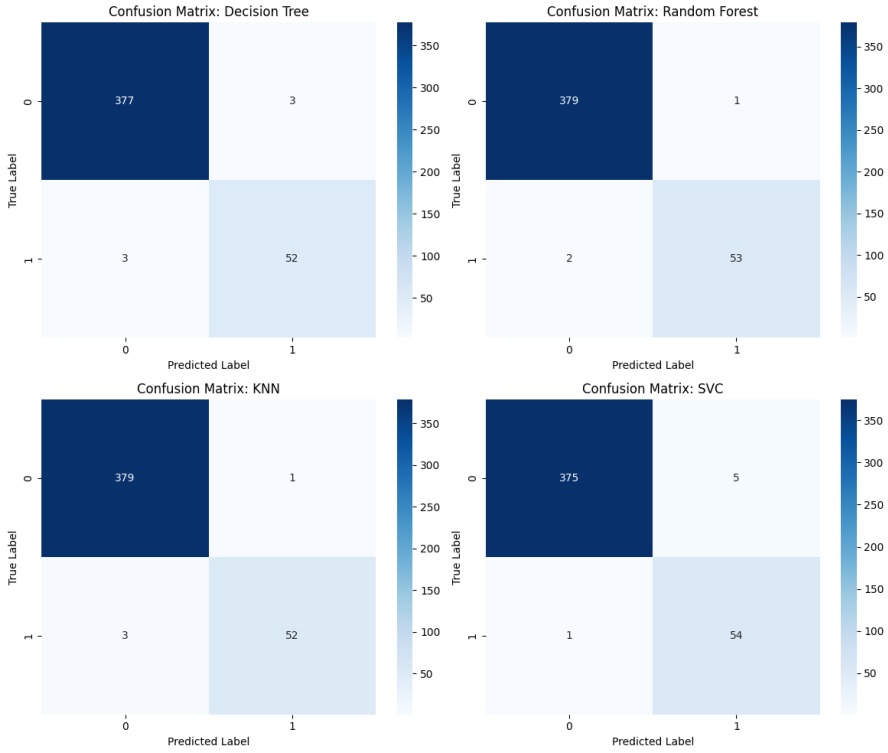

Figura 50 -Matrice di confusione - 1m attack - aggregation 5s - i=3 – undersampler

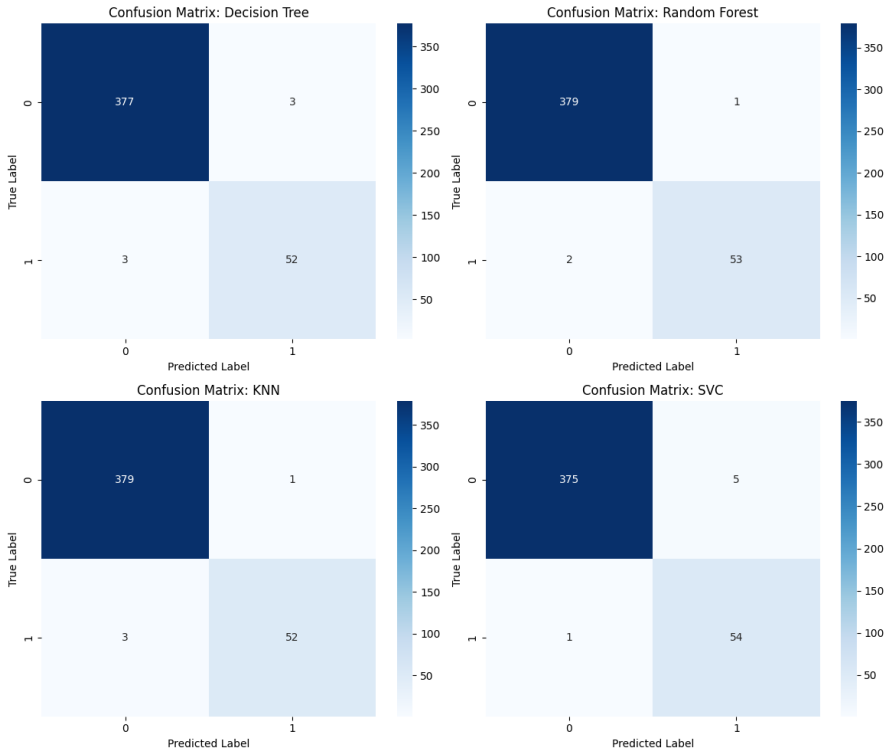

Figura 51 - Matrice di confusione - 1m attack - aggregation 5s - i=4 – undersampler

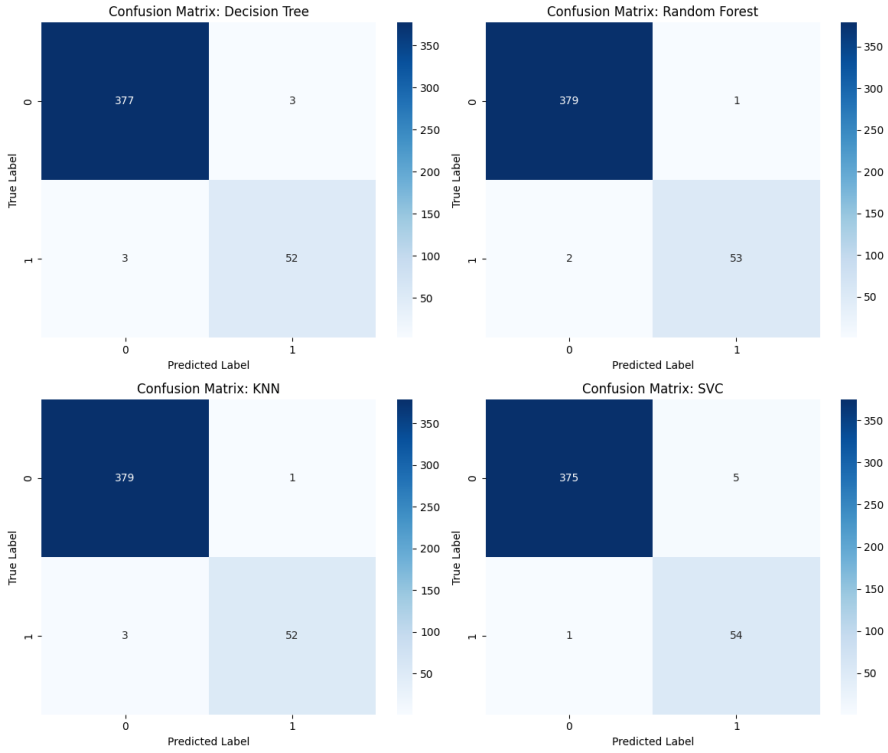

Figura 52 - Matrice di confusione - 1m attack - aggregation 5s - i=5 – undersampler

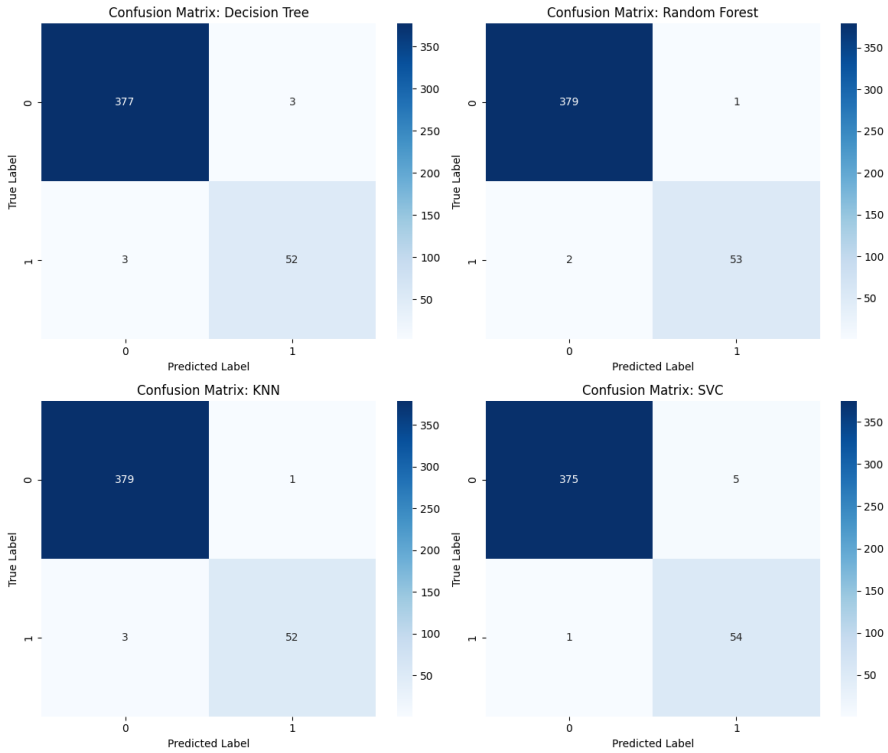

Figura 53 - Matrice di confusione - 1m attack - aggregation 5s - i=6 – undersampler

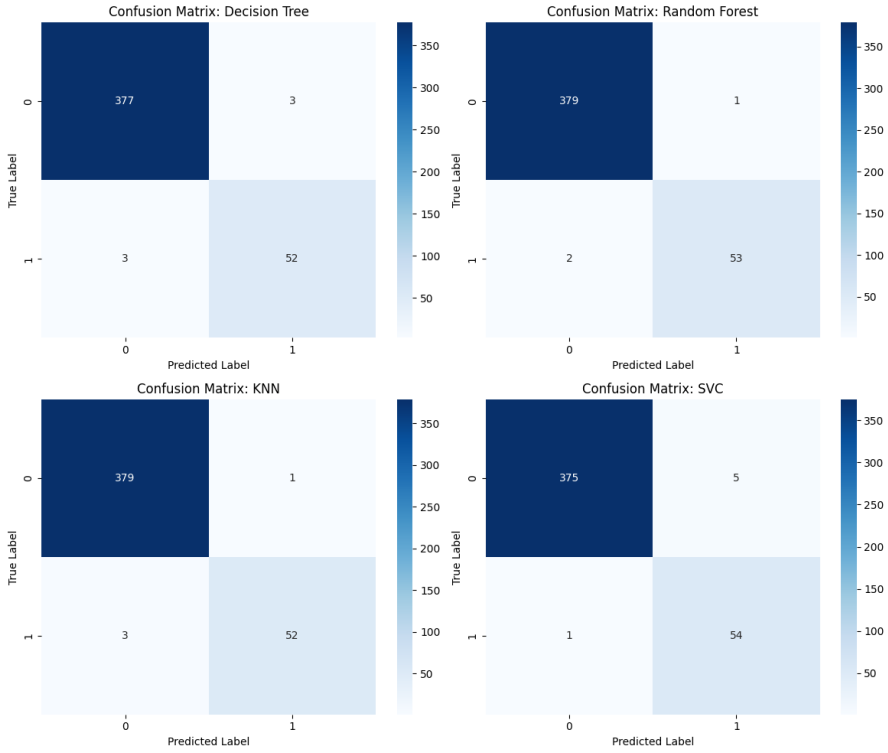

Figura 54 - Matrice di confusione - 1m attack - aggregation 5s - i=7 – undersampler

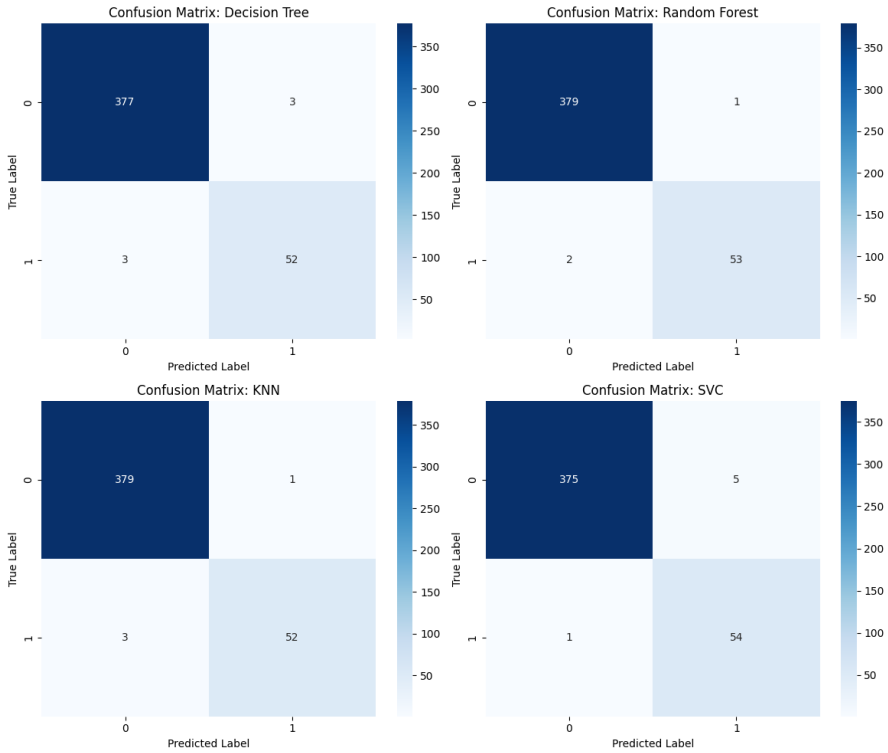

Figura 55 - Matrice di confusione - 1m attack - aggregation 5s - i=8 – undersampler

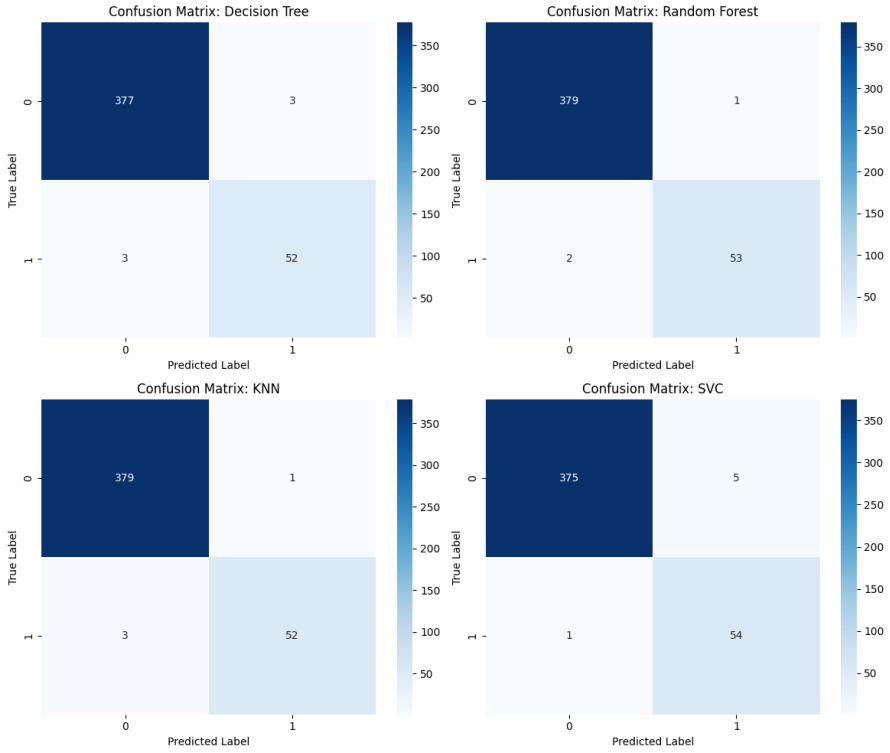

Figura 56 - Matrice di confusione - 1m attack - aggregation 5s - i=9 – undersampler

Nonostante la diminuzione del numero di catture malevole, i modelli continuano a classificare correttamente.

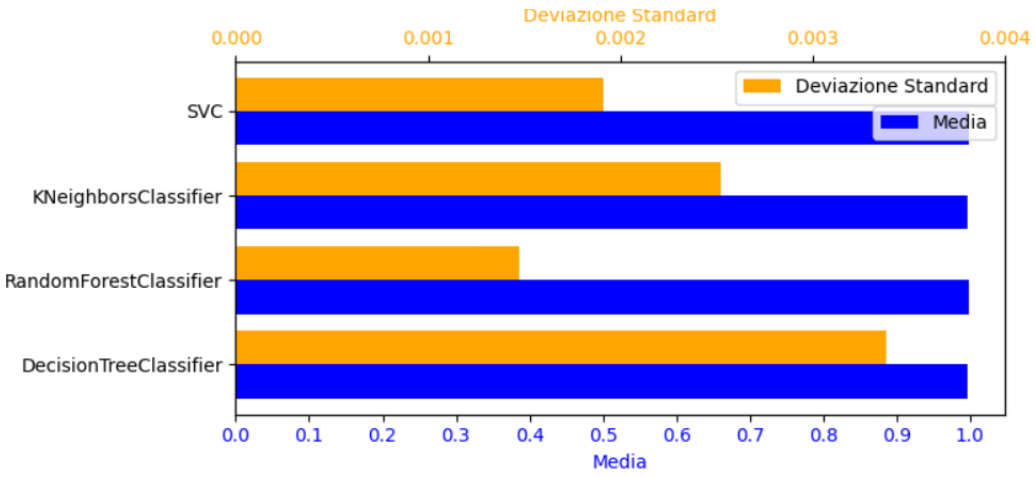

Figura 57 - Istogramma orizzontale - Media e deviazione standard accuracy – undersampler

Nella casistica in analisi, il Random Forest è quello che ottiene risultati più stabili, come illustrato dalla Figura 57.

CATTURA PING FLOODING 1M ATTACK - AGGREGATION 5S - SENZA UNDERSAMPLER

Le seguenti matrici di confusione sono state ottenute fornendo in input ai modelli il dataset sbilanciato contenenti catture malevole dalla durata complessiva di 1m.

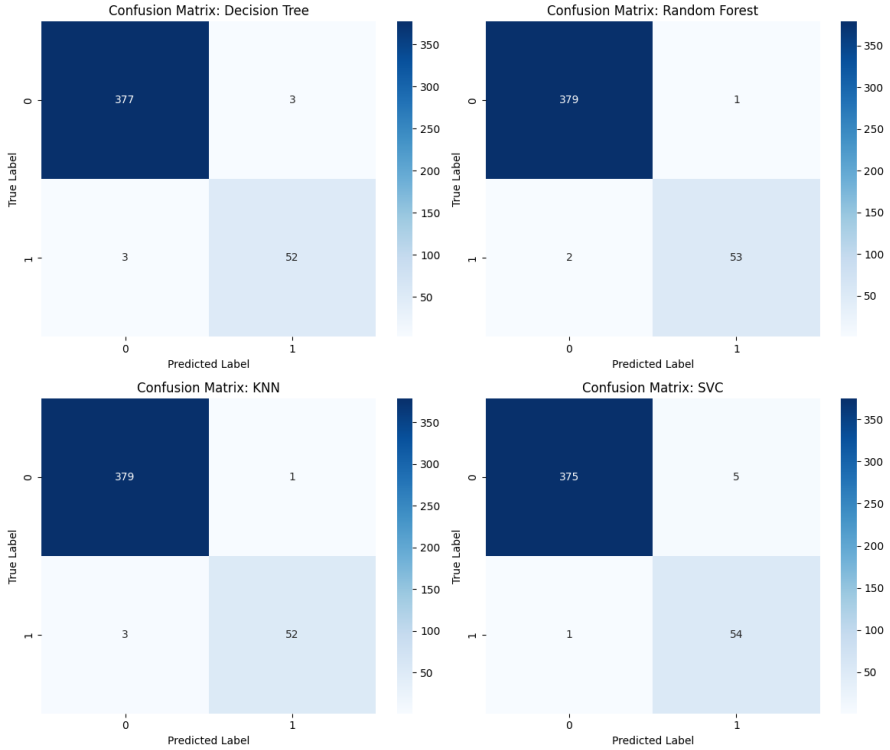

Figura 58 - Matrice di confusione - 1m attack - aggregation 5s - i=0 - no undersampler

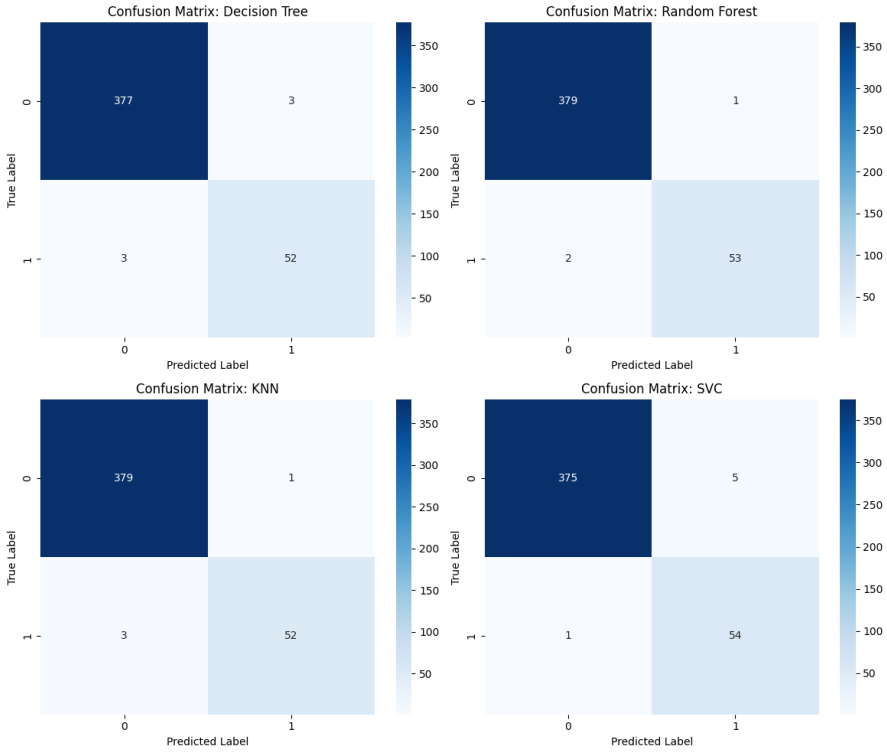

Figura 59 - Matrice di confusione - 1m attack - aggregation 5s - i=1 - no undersampler

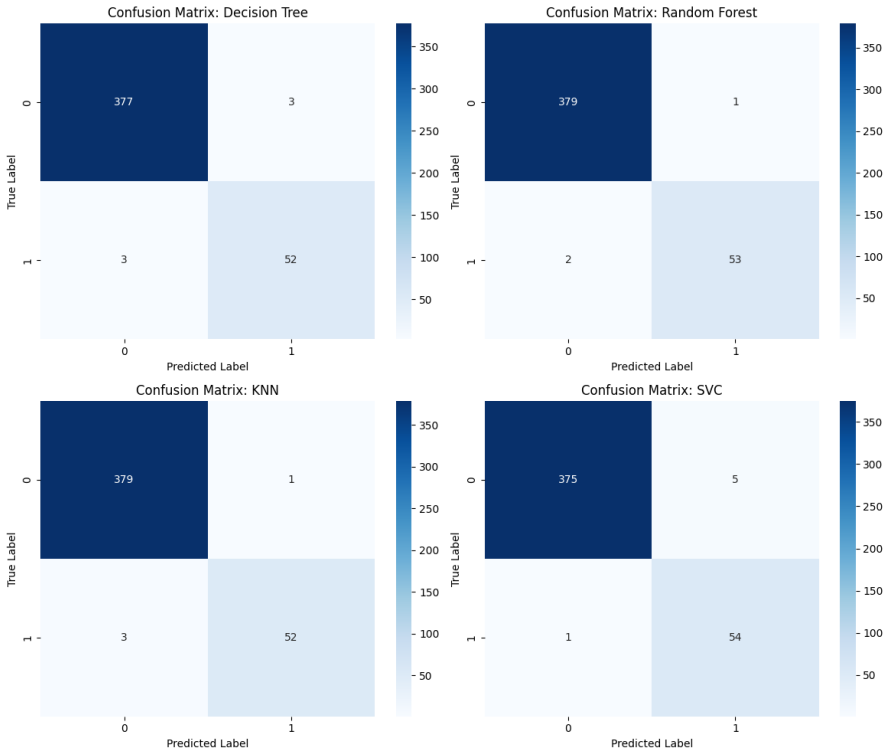

Figura 60 - Matrice di confusione - 1m attack - aggregation 5s - i=2 - no undersampler

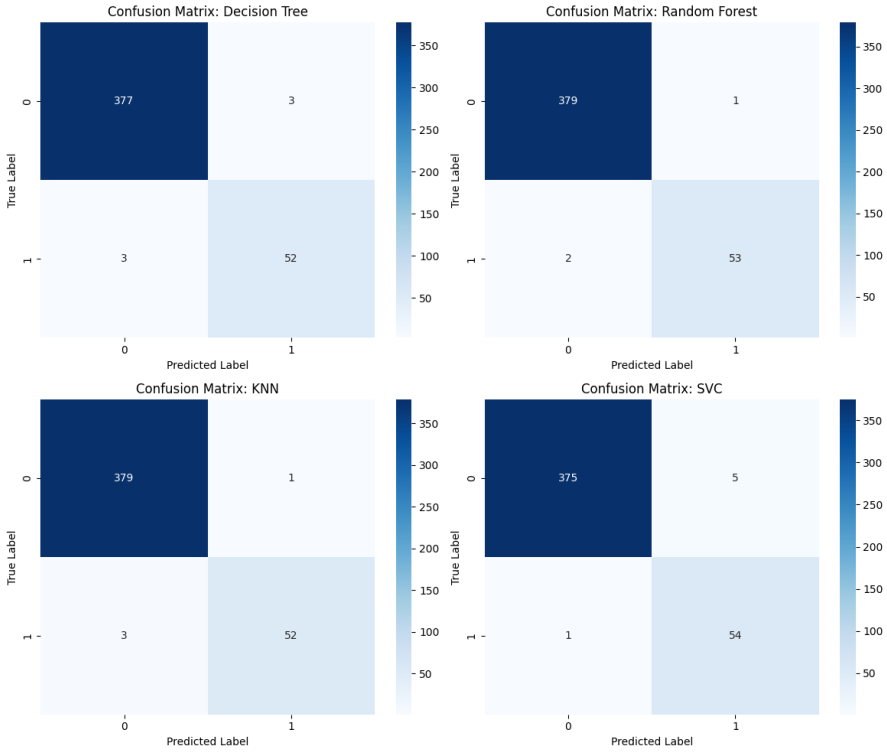

Figura 61 - Matrice di confusione - 1m attack - aggregation 5s - i=3 - no undersampler

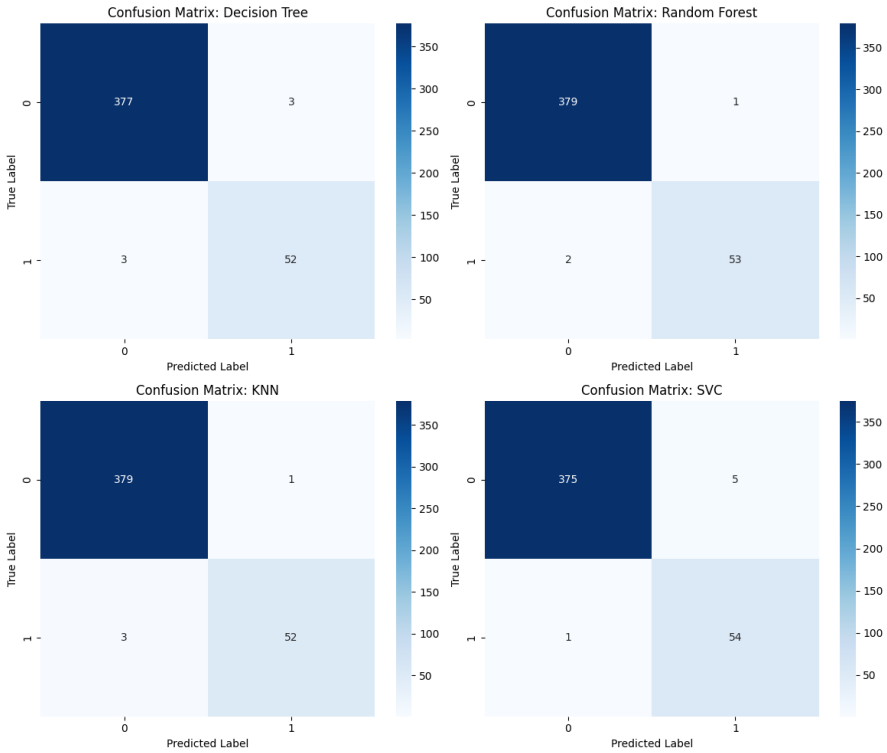

Figura 62 - Matrice di confusione - 1m attack - aggregation 5s - i=5 - no undersampler

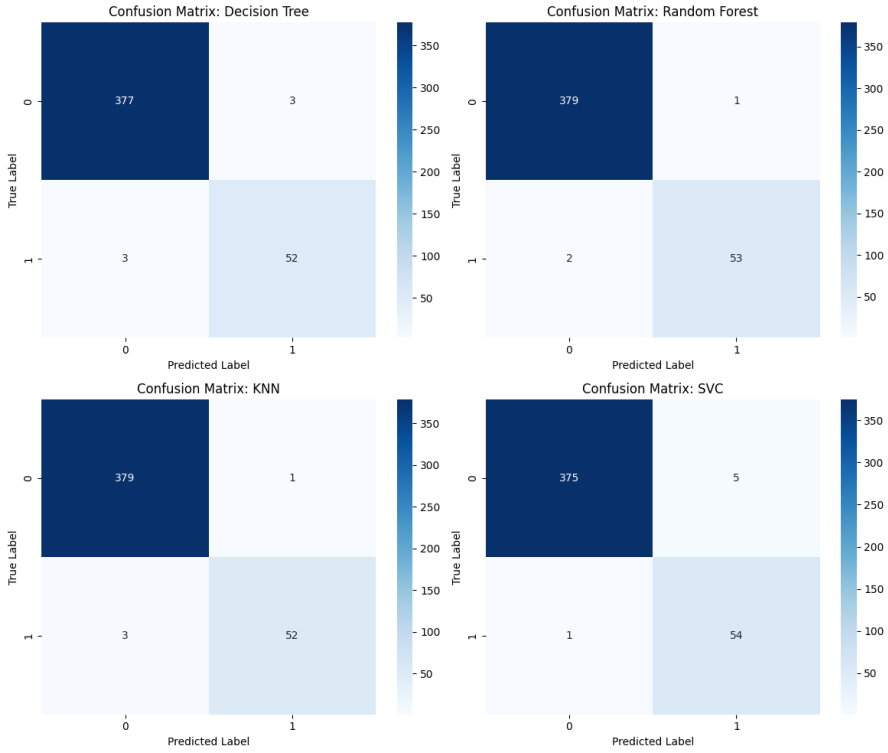

Figura 63 - Matrice di confusione - 1m attack - aggregation 5s - i=6 - no undersampler

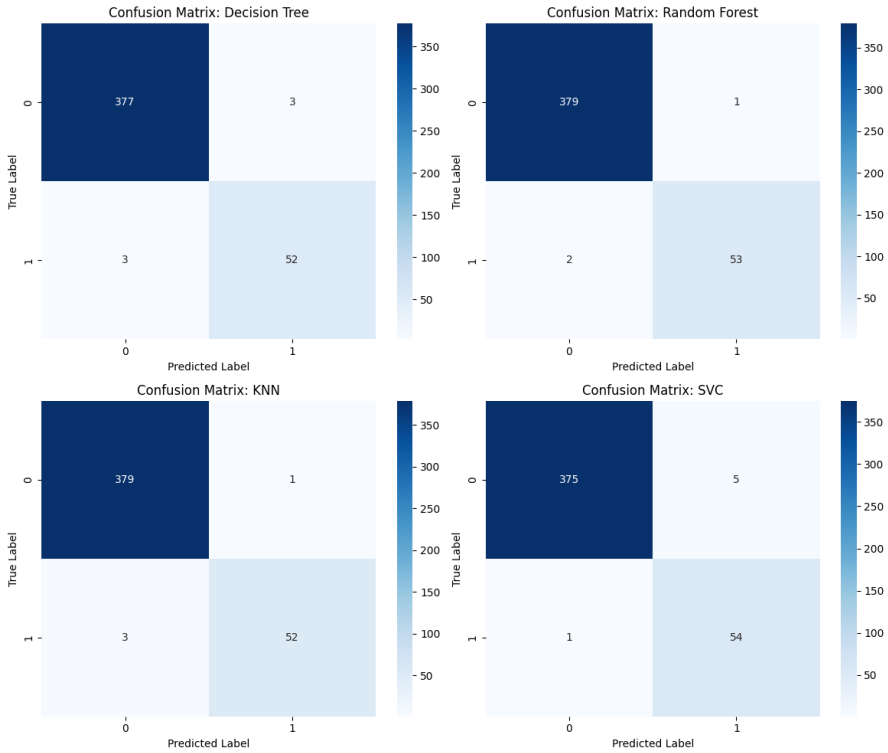

Figura 64 - Matrice di confusione - 1m attack - aggregation 5s - i=6 - no undersampler

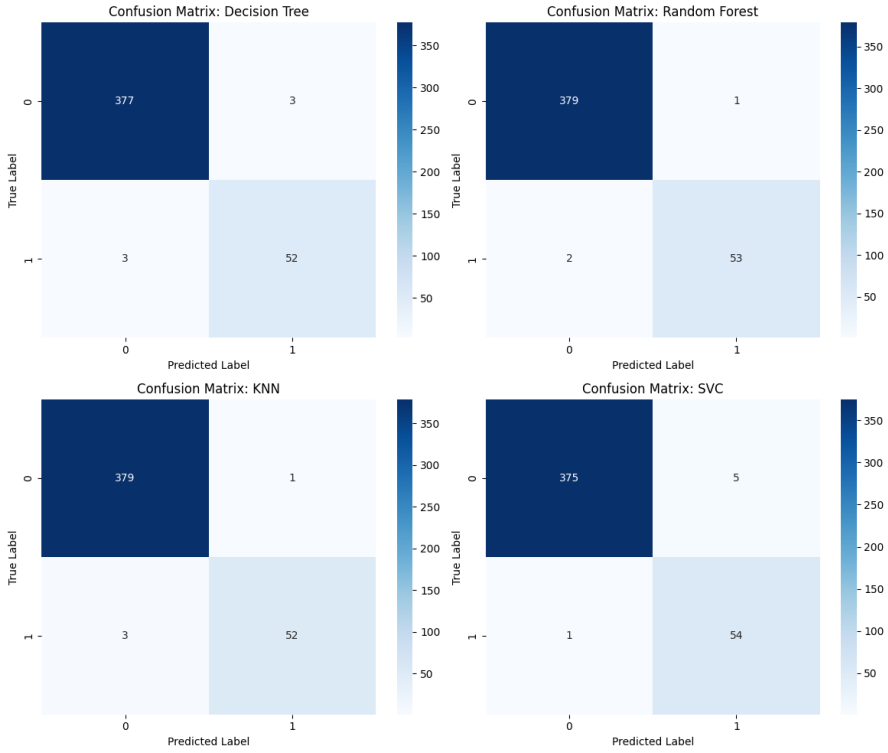

Figura 65 - Matrice di confusione - 1m attack - aggregation 5s - i=7 - no undersampler

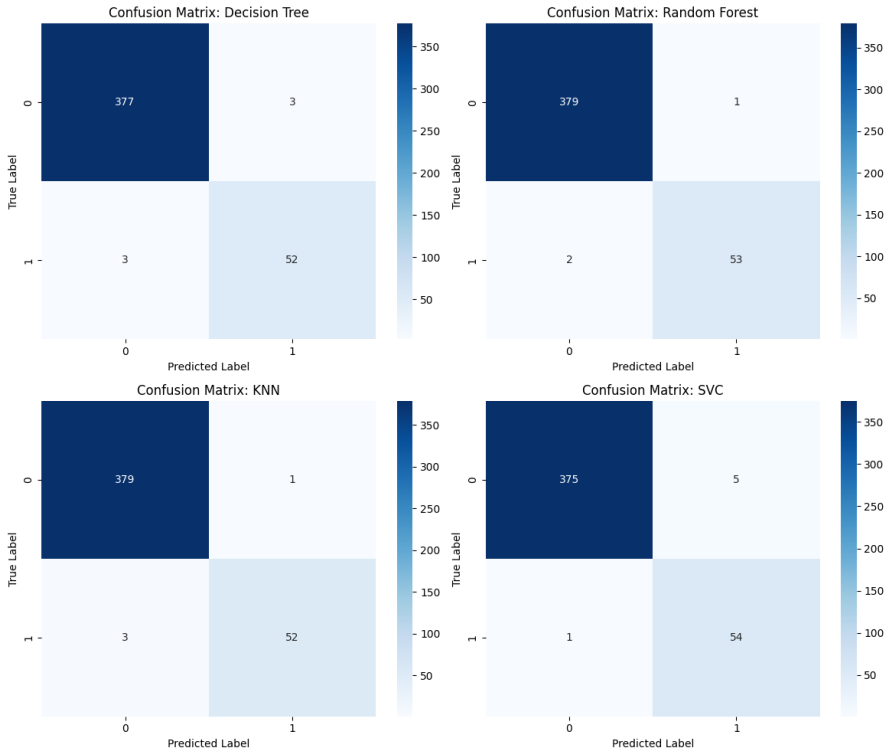

Figura 66 - Matrice di confusione - 1m attack - aggregation 5s - i=8 - no undersampler

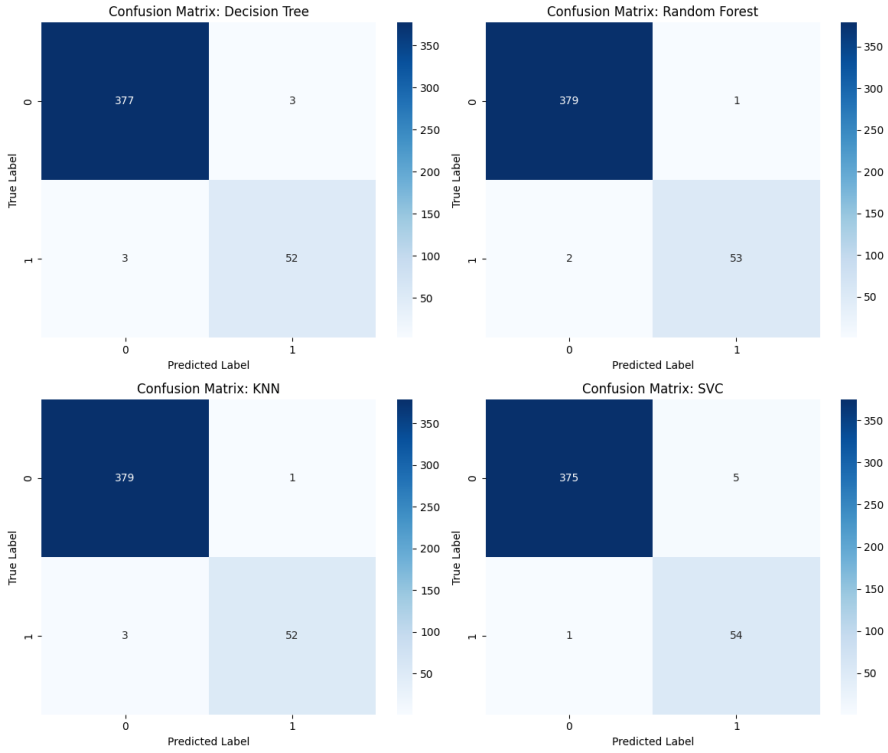

Figura 67 - Matrice di confusione - 1m attack - aggregation 5s - i=9 - no undersampler

Le matrici di confusione illustrano ancora una volta che lo sbilanciamento del dataset non influisce sulle prestazioni dei modelli.

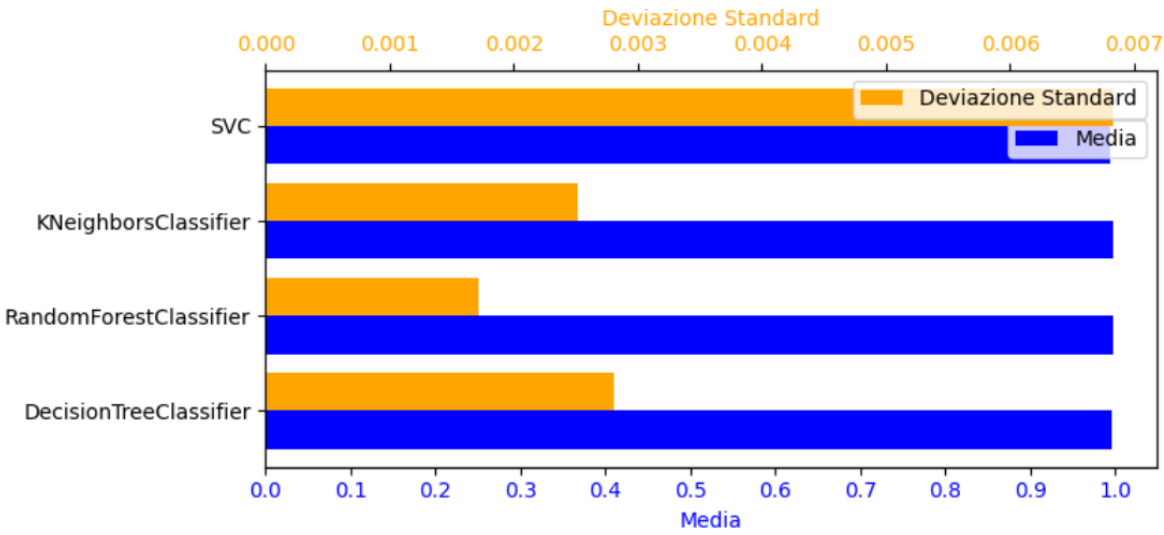

Figura 68 - Istogramma orizzontale - Media e deviazione standard accuracy - no undersampler

Tutti i modelli offrono ottime prestazioni, osservando la media delle diverse accuracy.
Seppure le deviazioni standard dell'accuracy sono molto piccole, il modello SVC in questo caso è quello che produce accuracy più variabili nelle varie iterazioni.

CONFRONTO COMPLESSIVO

Per fornire una panoramica generale delle prestazioni dei modelli si è deciso di produrre dei grafici riepilogativi che mostrano l'andamento delle accuracy medie dei modelli a seconda dei diversi tempi di attacco.
Confrontando le prestazioni dei modelli a cui è stato fornito in input un dataset bilanciato, si ottiene il grafico illustrato nell'immagine di seguito:

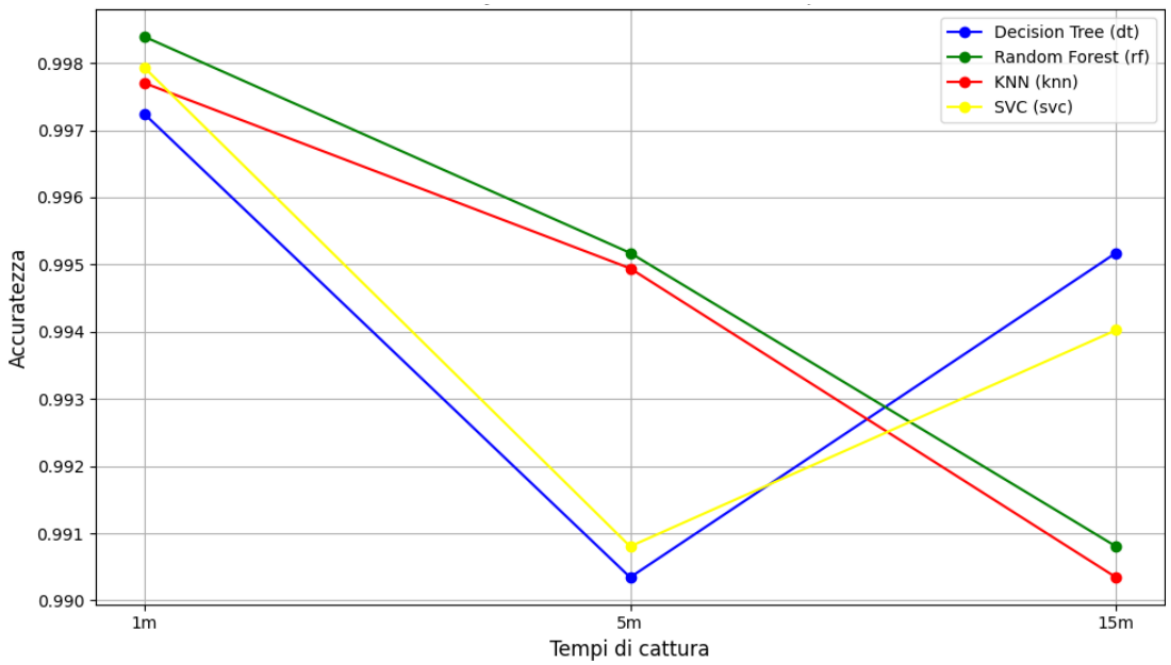

Figura 69 - Media Accuracy modelli – undersampler

Tutti i modelli forniscono risultati ottimi con durata del tempo di attacco pari ad 1 minuto nel caso della presenza dell'undersampler e quindi nel caso del dataset bilanciato.

I classificatori peggiorano le proprie previsioni nel caso di 5m di attacco, rispetto al caso di 1m.
Successivamente le performance di alcuni modelli incrementano mentre altre decrementano ulteriormente.
Nel paper di riferimento del lavoro probabilmente è stato utilizzato un dataset differente, quindi i risultati ottenuti non sono confrontabili.
Confrontando invece i diversi modelli a cui è stato fornito in input un dataset sbilanciato, si ottiene il grafico illustrato nell'immagine di seguito

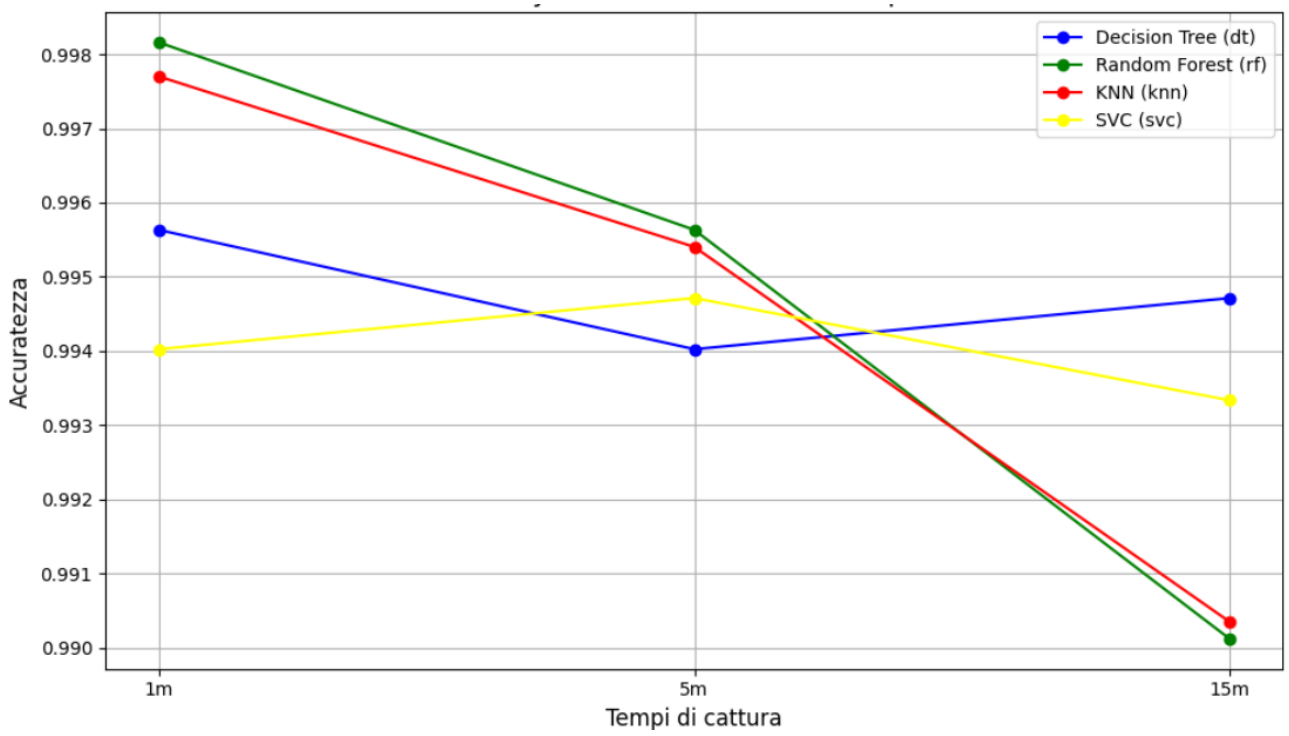

Figura 70 - Media Accuracy modelli – no undersampler

In questo caso, l'unico modello che sembra ricalcare quantomeno l'andamento illustrato nel paper è il Decision Tree.
Random Forest e KNN mostrano sempre un andamento decrescente mentre l'SVC ha il picco in termini prestazionali in corrispondenza del tempo di attacco pari a 5m.

CONCLUSIONI & SVILUPPI FUTURI

Aldilà dei confronti dei risultati con il paper di riferimento del lavoro, i modelli hanno fornito ottime prestazioni in tutte le casistiche.
Potrebbe essere interessante testarli su altre tipologie di attacchi DDoS, per verificare se possano essere così prestanti anche con essi.
Inoltre, si potrebbe esplorare approcci basati sul Deep Learning per l'ingegnerizzazione delle feature e la classificazione.
Dunque, si potrebbe verificare se si ottengono risultati equivalenti anche facendo dedurre le feature in maniera automatica al modello.