UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria

Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

Analisi e sviluppo di tecniche basate su Machine Learning per il rilevamento di intrusioni nelle reti OT

Analysis and development of Machine Learning-based techniques for intrusion detection in OT networks

Autore della tesi: **Simone Salvoni**

Relatore: **Prof. Marco Baldi**

Correlatore: **Giacomo Zonneveld**

Correlatore: **Lorenzo Principi**

A.A. 2022/2023

Sommario

Nella tesi qui presentata viene per prima cosa analizzato l'utilizzo degli Intrusion Detection System in ambito industriale. La sicurezza OT (Operational Techonology) è col tempo diventata un fattore sempre più critico, dato che un attacco informatico a una rete industriale può causare gravissimi danni non solo all'azienda colpita, ma anche a soggetti esterni, nei casi più gravi. Si pensi, per esempio, ai danni che possono essere causati da un attacco informatico diretto verso una rete di distribuzione energetica. Nella prima sezione della tesi sono dunque presentati gli IDS come possibile strumento di difesa, viene studiata la composizione e l'evoluzione dei sistemi SCADA e sono anche analizzati due dei protocolli di comunicazione maggiormente presenti in ambito industriale: il Modbus e l'S7comm.
Successivamente, nella seconda parte della tesi, sono invece presentati i risultati di esperimenti tramite i quali è stata testata la capacità di alcuni algoritmi di classificazione di distinguere fra traffico reale e sicuro e traffico malevolo all'interno di una rete di teleriscaldamento. Questi esperimenti sono preceduti dalla produzione di un dataset, le cui feature caratterizzano i singoli pacchetti trasmessi nella rete di riferimento in normale operatività, durante la quale sono state eseguite delle simulazioni di attacco. La produzione del dataset a partire da traffico reale di una rete industriale è un fattore determinante per la produzione di dati i più vicini possibile a una situazione di attacco reale, anche se questa scelta ha posto delle limitazioni sulle tipologie di attacco eseguibili. I classificatori testati sono di tre tipologie: classificatori supervisionati biclasse, classificatori supervisionati multiclasse e classificatori one-class. I risultati ottenuti nei vari esperimenti eseguiti sono promettenti, e mostrano una buona capacità di alcuni dei classificatori testati di gestire correttamente quasi tutte le classi di attacco presenti nel dataset.

Abstract

In the thesis herein, the use of Intrusion Detection Systems in the industrial environment is first analyzed. OT (Operational Techonology) security has over time become an increasingly critical factor, since a cyberattack on an industrial network can cause very serious damage not only to the affected company, but also to external parties in the most severe cases. Consider, for example, the damage that can be caused by a cyberattack directed at a power distribution network. Thus, in the first section of the thesis, IDSs are presented as a possible means of defense, the composition and evolution of SCADA systems are studied, and two of the communication protocols most commonly found in industrial environment are also analyzed: the Modbus and S7comm protocols.

Subsequently, in the second part of the thesis, a description of a few experiments, alongside their results, are presented. The ability of certain classification algorithms to distinguish between real, secure traffic and malicious traffic within a district heating network was tested through these experiments. These tests are preceded by the creation of a dataset, whose features characterize individual packets transmitted in the network under normal operation, during which attack simulations were performed. The production of the dataset from real traffic of an industrial network is a determining factor in producing data as close as possible to a real attack situation, although this choice placed limitations on the types of attacks that could be executed. The classifiers tested were of three types: supervised binary classifiers, supervised multiclass classifiers and one-class classifiers. The results obtained in the various performed experiments are promising, and they show a good ability of some of the tested classifiers to properly handle almost all attack classes present in the dataset.

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

Sono oramai anni che si assiste a una costante crescita delle problematiche relative alla *cybersecurity* in ogni ambito, sia lavorativo sia privato. Non è dunque sorprendente che sia di pari passo cresciuto l'interesse verso modalità di difesa implementabili per ridurre il più possibile le chance di essere colpiti da attacchi informatici.

Per quanto riguarda la difesa delle reti aziendali, oltre ai classici ma comunque importanti sistemi di difesa come i firewall, tra gli strumenti più efficaci che è possibile inserire nel proprio arsenale difensivo sono evidenziabili gli *intrusion detection system*, o IDS, ovvero i sistemi di rilevamento delle intrusioni. Gli IDS sono dei sistemi che monitorano la rete sulla quale sono posizionati e lanciano un allarme nel caso in cui rilevino un evento che ritengono pericoloso, in modo che poi possano essere prese le contromisure necessarie. Gli IDS sono quindi degli strumenti di monitoraggio passivo, che non vanno a interagire direttamente con il traffico di rete. Questo li differenzia dai cosiddetti *intrusion prevention system*, ovvero i sistemi di prevenzione delle intrusioni, che invece hanno il compito di agire attivamente sul traffico, identificando e bloccando tempestivamente azioni ritenute illecite.

Originariamente i sistemi industriali erano considerati sicuri rispetto alle classiche LAN/WLAN aziendali. Le reti dei sistemi industriali infatti per molto tempo erano separate da Internet, e questo le rendeva essenzialmente impenetrabili dall'esterno. Questo è anche il motivo per cui i principali protocolli di comunicazione degli ICS (sigla che sta per *industrial control system*), come *Profinet* o *Modbus*, non sono stati progettati tenendo in considerazione aspetti di sicurezza [5]. L'ipotesi di sicurezza dovuta all'isolamento è ormai decaduta da tempo a causa dell'introduzione della tecnologia IoT in ambito industriale. Tale tecnologia infatti richiede per il suo funzionamento un accesso stabile alla rete internet. Di conseguenza, le reti industriali che adottano l'IoT non possono più considerarsi sicure. Le problematiche che si sono così aperte non hanno una facile soluzione: i sistemi OT hanno caratteristiche e requisiti diversi rispetto a quelli IT, e dunque non è sufficiente implementare un IDS pensato per l'IT nell'OT. I sistemi OT infatti necessitano di requisiti di prestazioni molto più stringenti rispetto ai sistemi IT. Per esempio, si impongono limiti stretti ai tempi di trasmissione massimi dei pacchetti da parte dei sistemi di controllo delle macchine. Ciò rende l'attività di monitoraggio della rete una sfida critica che richiede così un attento bilanciamento tra l'efficienza operativa delle macchine e la capacità di analisi e rilevamento delle minacce. Un'altra differenza estremamente importante è che la tipologia di attacchi è diversa dalle reti comuni, sia per obiettivi sia per modalità. Come è sottolineato in [5], per esempio, un potenziale attacco molto grave per un sistema industriale (che non avrebbe senso vedere in normali sistemi IT) è la modifica di parametri di lavoro delle macchine, oppure dei loro tempi di esecuzione. Entrambe le modifiche infatti possono causare enormi danni.

1.1 Obiettivi e struttura della tesi

Questa tesi ha due scopi principali. Per prima cosa verrà presentata un'analisi dello stato dell'arte degli *Intrusion Detection System*, con un focus particolare sul loro utilizzo nell'ambito OT. Successivamente, verrà presentato il lavoro svolto per la creazione di un dataset, che è stato poi utilizzato per il test di vari algoritmi di classificazione nell'ambito del rilevamento delle intrusioni. Lo scopo principale di questi test di classificazione è quello di studiare le capacità di vari algoritmi di classificazione nell'ambito del rilevamento intrusioni su traffico industriale, con focus sui protocolli Modbus e S7comm, in due condizioni specifiche:

1. Il dataset utilizzato per l'allenamento viene costruito a partire da traffico reale di una rete attivamente in produzione.
2. Le feature estratte dal dataset vanno a caratterizzare il singolo pacchetto trasmesso.

La tesi è strutturata come segue:

- Nella sezione 2 verrà presentata un'analisi riguardante gli *Intrusion Detection System*: le loro tipologie, le caratteristiche da ricercare e le tecniche per la loro costruzione.
- Nella sezione 3 verrà data una descrizione dei sistemi SCADA, di come sono generalmente composti, di come sono evoluti nel tempo e a quali tipologie di attacchi sono tipicamente vulnerabili.
- Nella sezione 4 verrà analizzato lo stato della ricerca scientifica nel campo del rilevamento intrusione su reti OT e la situazione che si incontra nell'ottenimento di dataset pubblici.
- Nella sezione 5 verrà discussa la metodologia di creazione del dataset.
- Nella sezione 6 verranno presentati i test di classificazione e i loro risultati.
- Infine, nella sezione 7 saranno presentate le conclusioni finali e verranno proposti possibili avanzamenti e miglioramenti.

2 IDS

2.1 Tipologie di IDS

Come ben presentato in [2], è possibile suddividere gli IDS secondo due diversi criteri. Il primo di tali criteri si basa sulle informazioni che gli IDS utilizzano per lavorare. Su tale base, sono definibili tre categorie di IDS:

- ***Network-based* IDS, o NIDS**. Questa tipologia di IDS studia il traffico della rete che monitora per rilevare possibili attacchi.
- ***Host-based* IDS**. Questi IDS studiano i log degli host della rete, per ricercare eventuali eventi anomali.
- **IDS ibridi**, che sfruttano entrambe le fonti.

Come evidenziato in [36], l'approccio idealmente da preferire è quello ibrido, in quanto il monitoraggio sia del traffico di rete sia degli host permette il rilevamento di un maggior numero di tipologie di attacco, il che a sua volta permette di avere uno sguardo più ampio su tutta l'infrastruttura. Di contro però, l'IDS non solo sarà più complesso da realizzare, ma sarà anche soggetto a carichi di lavoro molto più pesanti a causa dell'elevato flusso di informazioni da processare. Quest'ultimo aspetto, come già evidenziato, risulta particolarmente critico nel contesto di reti industriali dove è richiesta una comunicazione tempestiva tra le macchine. Per questi motivi, a dipendenza dell'architettura con cui si lavora, un IDS più leggero potrebbe essere preferibile.
Un secondo criterio per classificare gli IDS invece si basa sulle metodologie di monitoraggio che l'IDS sfrutta. Anche in questo caso possiamo suddividere gli IDS in tre gruppi:

- ***Signature-based* IDS**, ovvero IDS basati su firma. Con "firma" si intende un pattern riconoscibile e riconducibile a un attacco. Questi IDS si basano su delle regole costruite per identificare queste firme. Se, durante il monitoraggio, una di queste regole viene soddisfatta l'IDS lancia un allarme.
- ***Behavioral* IDS**, ovvero IDS comportamentali. Questi IDS hanno l'obiettivo di definire un profilo comportamentale standard dell'entità monitorata, in modo tale da poter identificare comportamenti anomali interpretabili come potenziali sintomi di minacce informatiche in atto (ma non solo, dato che un'anomalia può anche avere altre fonti). Per questo questa categoria di IDS è anche detta quella degli *Anomaly-based* IDS.
- **IDS ibridi**. In questo caso la natura ibrida degli IDS è definita dal loro utilizzare contemporaneamente entrambi gli approcci precedenti, spesso in cascata.

Anche in questo caso, l'approccio più adatto da seguire per un IDS non è ovvio. Quelli basati su firma, attualmente, sono quelli che forniscono tendenzialmente le performance migliori a livello di accuratezza e di tasso di falsi positivi/negativi. Tuttavia, l'approccio a firma ha una grossa

limitazione: l'incapacità di rilevare attacchi per cui non sono definite delle regole. Questa situazione potrebbe essere sia il risultato di un errore umano (in nessun caso un elemento ignorabile nel campo della sicurezza informatica), sia una conseguenza dell'effettiva inesistenza di firme di riconoscimento associabili a un particolare attacco. Questo è particolarmente vero per gli attacchi *zero-day*. La questione degli *zero-day* è particolarmente sentita, ed è per questo che c'è un enorme interesse nei confronti dell'approccio comportamentale. La ricerca, attualmente, si sta infatti concentrando specialmente su questa tipologia, e soprattutto sull'adozione di tecniche basate sul machine learning. Tuttavia, all'atto pratico, tuttora gli approcci basati su machine learning non hanno fornito ancora dei risultati paragonabili agli IDS basati su firma, specialmente per via di un alto tasso di falsi positivi e negativi che ancora presentano.

2.2 Caratteristiche di un buon IDS

Un IDS dovrebbe possedere varie caratteristiche per potersi considerare utilizzabile. Oltre ad accuratezza e precisione, devono infatti essere presi in considerazione altri aspetti rilevanti per garantire l'applicabilità in pratica di tale strumento. In [2] vengono definite sei caratteristiche indispensabili per un buon IDS, identificate con l'acronimo CLARRA, e altre quattro che sono invece non obbligatorie ma auspicabili, identificate con l'acronimo HISS.
Le caratteristiche "CLARRA" sono:

- **Feedback continuo (*continuous feedback*)**: un IDS deve essere capace di migliorare nel tempo, imparando dai propri errori. Per esempio, un IDS basato su firma dovrebbe poter integrare nuove firme per aggiornarsi, mentre un IDS basato su una rete neurale dovrebbe periodicamente essere ri-allenato con dei dataset più aggiornati.

- **Bassi tassi di falsi positivi e negativi (*low false alarms/positive and negative rates*)**. I falsi positivi e negativi (ovvero falsi allarmi e mancate rilevazioni) sono particolarmente gravi nel campo del monitoraggio delle reti, per chiare ragioni. Nel primo caso si rischia di causare disagi al traffico lecito, mentre nel secondo si lascia passare traffico illecito, con tutte le conseguenze del caso. La questione è ancora più importante nell'OT, dove il blocco di traffico lecito potrebbe causare serie problematiche al lavoro dei macchinari.

- **Rilevamento accurato (*accurate detection*)**. Un IDS deve, ovviamente, avere un'alta accuratezza nel riconoscimento delle anomalie.

- **Reattività (*responsivness*)**. Un buon IDS deve restituire i propri risultati in tempi sufficientemente brevi, per permettere di prendere contromisure tempestive se necessario.

- **Resilienza (*resilience*)**. Un IDS deve essere resistente agli attacchi che potrebbero tentare di manomettere il suo funzionamento. Se così non fosse, un IDS altrimenti molto performante potrebbe essere reso facilmente inutile.

- **Agilità (*agility*)**. L'IDS deve essere sufficientemente leggero a livello di costo computazionale. Un carico computazionale troppo elevato potrebbe portare a un rallentamento della rete, particolarmente problematico poi per una rete OT.

Le caratteristiche "HISS" invece sono le seguenti:

- **Ibridazione (*hybrid*)**. Un buon IDS dovrebbe implementare sia una componente basata su firma, sia una componente comportamentale, per essere il più meticoloso possibile.
- **Interattività (*interactive*)**. Un buon IDS dovrebbe comunicare con gli amministratori di sistema in maniera semplice e comprensibile, e dovrebbe possedere un buon sistema di interrogazione, per permettere una facile estrazione di informazioni.
- **Difesa (*shielded*)**. Un buon IDS dovrebbe mettere in atto delle contromisure, dove possibile, per bloccare il traffico che evidenzia come malevolo.
- **Specificità (*specific*)**. Un buon IDS dovrebbe essere capace riconoscere la tipologia di minaccia che rileva.

2.3 Approcci per lo sviluppo di IDS

In questa sezione verranno analizzate più nel dettaglio le due categorie principali di IDS, già presentate nel paragrafo 2.2, ovvero gli IDS a firma e quelli comportamentali. Tramite un'analisi dei principi e degli strumenti alla base del loro funzionamento saranno confrontati i loro pro e i loro contro.

2.3.1 IDS a firma

Come precedentemente sottolineato, gli approcci basati su firma sono quelli che forniscono le performance più solide, e sono quindi anche quelli più comunemente usati all'atto pratico. Tendenzialmente, per costruire IDS di questa tipologia ci si basa su strumenti open source che permettono liberamente di definire regole, in modo tale da personalizzare il sistema di monitoraggio per la propria specifica rete. I due strumenti più comunemente utilizzati per gli IDS sono, secondo [5]:

- **Snort** [43]. Il più popolare, con la community più attiva. Sono disponibili dei pacchetti di regole già pronte specificatamente per l'OT.
- **Suricata** [45]. Soluzione simile a Snort, compatibile con il formato delle sue regole. È più complicato ed ha una community meno attiva, tuttavia permette di sfruttare un approccio multithread, che potrebbe essere un vantaggio

Si menziona anche **Zeek** (un tempo chiamato Bro) [47], una soluzione alternativa che lavora diversamente da Snort e Suricata e che quindi potrebbe comportarsi meglio o peggio a seconda delle situazioni. Zeek è in realtà un IDS che permette anche di lavorare con approccio *anomaly detection*, oltre che con firme di attacchi. Sono poi presenti anche soluzioni commerciali, con aziende che offrono soluzioni più complesse.
È importante che questa tipologia di IDS venga costantemente aggiornata per includere le firme di attacchi più recenti e rilevanti. Una tecnica potenzialmente molto utile per la produzione di nuove regole è lo sfruttamento dagli *honeypot*, come viene evidenziato dall'articolo [46]. Costruire sistemi volontariamente vulnerabili per attrarre attaccanti può risultare un ottimo modo di ottenere dati

su attacchi reali da introdurre poi nel proprio IDS. Ovviamente non è un approccio semplice da seguire, dato che costruire un buon honeypot non è banale.
Gli approcci basati su firma sono quindi ancora molto efficaci e utilizzati. Tuttavia, come già menzionato, hanno vari limiti che hanno portato la ricerca verso lo studio di opzioni alternative. Uno di questi limiti è quello già descritto in precedenza, ovvero l'incapacità totale di gestire gli attacchi *zero-day*, e in generale ogni attacco per cui non è stata prodotta una firma. Un altro limite non trascurabile inoltre riguarda l'incapacità di produrre firme per alcune tipologie di attacco, specialmente in ambito OT. Se ne trova un esempio nell'articolo [14], dove gli autori menzionano come un possibile attacco nelle reti OT (i quali sono più ampiamente analizzati nella sezione 3.4) l'iniezione di comandi falsi, ma molto simili se non identici, a comandi legittimi. Questa *command injection* è particolarmente problematica dato che il problema non sta nel suo contenuto, quanto nel suo tempismo. Si pensi, per esempio, a un comando di reset richiesto ed eseguito nel momento sbagliato. In questi casi definire una firma per l'attacco è molto difficile, dato che questi comandi sarebbero completamente legittimi e corretti nel giusto contesto.
Per questi motivi quindi c'è un grande interesse nello sviluppo di IDS che seguono approcci diversi, primo fra tutti quello comportamentale, descritto nella sezione seguente.

2.3.2 IDS comportamentali

Per quanto riguarda invece gli IDS di tipo comportamentale, gli approcci testati dalla letteratura sono molteplici. Sono evidenziabili approcci di tipo statistico-probabilistico, che hanno solide basi teoriche ma tendono a essere difficili da ideare e ancor di più da costruire. Ultimamente c'è molto interesse per un approccio che sfrutta la teoria dei grafi. Questo approccio si basa su solide teorie matematiche, ma, come è evidenziato in [2], ci sono dei problemi non da poco da affrontare. Prima di tutto, non è ancora chiaro quale tecnica *graph-based* sia la più adatta da implementare. L'altro grande problema è che la complessità computazionale dei problemi tende a crescere troppo, anche per via del fatto che le dimensioni dei grafi tendono a esplodere con la crescita della rete.
L'approccio sicuramente più comune e studiato è però quello basato sul machine learning. Le motivazioni sono ovvie: il machine learning è ampiamente sfruttato in tantissimi campi diversi, con ottimi risultati. Algoritmi come gli alberi decisionali e le reti neurali si sono dimostrati molto efficienti nel risolvere problemi di *patter matching*. Tuttavia il problema di classificazione trattato dagli IDS è particolarmente complesso, e gli algoritmi di machine learning hanno ancora difficoltà a raggiungere le performance che presentano in altri contesti, specialmente nel caso di predizioni multiclasse. Nello specifico, le problematiche più grosse che si incontrano attualmente sono i tassi troppo alti di falsi positivi e negativi, che, come precedentemente analizzato, sono un grosso problema nel campo degli IDS.
Gli approcci supervisionati sono quelli più comunemente studiati e utilizzati, dato che sono quelli più conosciuti e che forniscono tipicamente risultati migliori rispetto alle controparti non supervisionate. Le famiglie di classificatori ad apprendimento supervisionato che attualmente risultano più comuni a livello di ricerca sono:

- **Le reti neurali**. Sono testati vari approcci, con reti feed-forward, convoluzionali ed LSTM.

Verso le LSTM c'è particolare interesse, grazie alla loro capacità di riconoscere pattern nei dati nel tempo, cosa potrebbe risultare particolarmente utile per analizzare una sequenza di pacchetti.

- **Le random forest**. Tra gli approcci ad *ensemble* le random forest sembrano performare particolarmente bene in questo ambito
- Le **SVM, o Support Vector Machine**.

Le tecniche non supervisionate hanno invece attirato molto interesse perché, non richiedendo una fase di allenamento basata su un dataset pre-etichettato, sono, a livello teorico, gli algoritmi che potrebbero primeggiare nel riconoscimento di nuove minacce [44]. Tuttavia, è importante sottolineare che i risultati ottenuti sono, allo stato attuale, generalmente peggiori di quelli ottenuti dagli algoritmi supervisionati. Secondo [2] gli algoritmi non supervisonati che ultimamente compaiono maggiormente in letteratura sono il K-means e la SOM, o *Self Organizing Map*.
Sebbene non allo stesso livello delle categorie precedenti, in letteratura sono anche riscontrabili IDS comportamentali costruiti con un approccio ad apprendimento per rinforzo. Per questa categoria di algoritmi di classificazione è presente una fase di allenamento il cui scopo è la massimizzazione di una certa funzione obiettivo. Un algoritmo particolarmente sfruttato è il *Q-learning*.
Nonostante il grande interesse e i risultati promettenti a livello di ricerca, come già sottolineato le performance dell'approccio comportamentale non sono ancora riuscite a essere paragonabili a quelle che si ottengono con gli IDS a firma. Le cause sono da ricercarsi in un insieme di problematiche diverse. Una delle maggiori, che colpisce specificatamente l'approccio supervisionato e che risulta particolarmente accentuato nel campo OT, è quella della carenza di adeguati dataset. I dataset più comunemente usati in ambito di ricerca sono quelli liberamente accessibili, poiché presentano il grande vantaggio di essere pronti all'uso. Purtroppo però risultano spesso poco aggiornati e limitati nelle tipologie di attacco presenti. Inoltre, peccano per tutti i limiti derivanti dalla rappresentazione di un caso d'uso generico che non sempre può essere adattato al caso particolare per cui si realizza un IDS. D'altra parte, la costruzione di un dataset ad hoc comporta il dover affrontare problemi spesso analoghi a quelli riscontrati con i dataset pubblici, quali la limitata rappresentanza di comportamenti malevoli. Inoltre risulta spesso necessaria l'esecuzione di operazioni di pulizia dei dati grezzi raccolti e di feature selection. Di conseguenza, se tali problemi non vengono gestiti correttamente, si rischia di ottenere predittori affetti da underfitting o da overfitting. Soprattutto quest'ultimo problema è particolarmente insidioso, dato che può risultare difficile riconoscerne la presenza, a maggior ragione in mancanza di buoni dati alternativi su cui testare le performance.
Riguardo l'efficacia di rilevamento di *zero-day attack* da parte di algoritmi supervisionati c'è poi ancora qualche dubbio, come è sottolineato in [5], dato che essi si allenano su dataset che contengono solo attacchi noti. Per questo motivo, nonostante il peggioramento delle prestazioni che tipicamente si incontra, si potrebbe preferire l'uso di algoritmi non supervisionati, in quanto non richiedono per l'allenamento un dataset contenente sia il comportamento normale che il malevolo. Da parte di alcuni c'è più fiducia sulle loro capacità di riconoscere anomalie, e quindi attacchi, mai viste prima. Nonostante tutto però, l'idea comune è che l'approccio comportamentale sia ancora molto promettente.

3 Sistemi SCADA

In questa sezione verranno analizzati i sistemi SCADA: cosa sono, come sono composti generalmente e come si sono evoluti nel tempo, in modo da averne una comprensione più ampia che permetta di analizzare con maggior precisione le vulnerabilità di questi sistemi. Questa analisi è supportata dalle descrizioni presenti nell'articolo [1]. Successivamente, verranno analizzati le tipologie di attacco più comuni che possono colpire le reti industriali, per poter comprendere al meglio da cosa è importante difendersi.

3.1 Componenti di un sistema SCADA

I sistemi SCADA, sigla che sta per *supervisory control and data acquisition*, sono sistemi di raccolta ed elaborazione di informazioni per la gestione di sistemi fisici. Sono composti tendenzialmente da quattro classi di componenti principali:

- **Componenti di supervisione**. Queste si presentano nella forma di HMI, o *Human Machine Interface*, e MTU, o *Master Terminal Unit*. Le prime sono interfacce di supervisione che presentano informazioni rilevanti sullo stato del sistema agli esperti di settore, che devono monitorare l'andamento del lavoro. Le seconde invece sono sistemi centralizzati che comunicano con sistemi di livello più basso, come PLC e RTU, presentati nel punto successivo

- **Componenti di acquisizione dati**. In questo caso si parla di PLC, o *Programmable Logic Controller*, e RTU, ovvero *Remote Terminal Unit*. Entrambi raccolgono i dati sui singoli dispositivi ottenuti da sensori, li elaborano sulla base della loro logica, e spediscono i risultati ai sistemi di supervisione. La differenza principale fra le due categorie è che i PLC sono più adatti al controllo locale, mentre le RTU sono pensate per gestire aree geograficamente più vaste, dato che funzionano tramite comunicazioni wireless.

- **Componenti di Data Storage**. I dati raccolti devono chiaramente essere salvati permanentemente. Tendenzialmente i sistemi SCADA sfruttano database relazionali classici, interrogabili tramite query SQL. Un software tipico per gestire questa sezione è *Historian*.

- **Componenti di scambio dati**. Sono rappresentati dall'infrastruttura di rete e dai protocolli di comunicazione utilizzati. Tra i protocolli più tipici troviamo Modbus, S7Comm, Profibus, Profinet e DNP3.

Oltre a questi sono ovviamente presenti i singoli device del sistema industriale, che il sistema SCADA controlla.

3.2 Evoluzione dei sistemi SCADA

I sistemi SCADA si sono evoluti nel tempo, con un accrescimento della complessità e della connettività. Questo ha comportato un significativo aumento delle loro potenzialità, ma al contempo

ha introdotto nuove problematiche da affrontare. È possibile evidenziare quattro "generazioni" di sistemi SCADA.

- **Sistemi SCADA monolitici**. Originariamente i sistemi SCADA erano basati su un'architettura monolitica e centralizzata, che ruotava attorno a un unico mainframe ad alte capacità computazionali, connesso in WAN con le RTU o in LAN con i PLC. I protocolli di comunicazione usati erano tendenzialmente proprietari dei dispositivi di acquisizione e gestione dati utilizzati. Spesso era presente inoltre un mainframe secondario in standby, da utilizzare in caso di guasti. Questi sistemi erano completamente isolati dall'esterno, e quindi sicuramente molto più protetti da minacce esterne rispetto ai sistemi moderni. Tuttavia, per migliorare l'efficienza e ridurre i costi, un'evoluzione verso architetture distribuite era inevitabile.

- **Sistemi SCADA distribuiti**. Con il tempo i sistemi si sono evoluti per dividere il lavoro che un tempo era gestito esclusivamente dal dispositivo centrale fra più dispositivi distribuiti connessi in una LAN. Suddividere i vari aspetti del lavoro in più unità elaborative aumenta l'affidabilità del sistema. Analogamente, con la maggiore distribuzione aumenta la resilienza ai guasti.

- **Sistemi SCADA *networked***. Questa tipologia di architettura è molto simile alla precedente, ma con una differenza sostanziale: la sostituzione dei protocolli proprietari con protocolli standard comunemente utilizzati finora solo nell'ambito IT. Nello specifico, compaiono il protocollo IP per la comunicazione fra RTU e stazioni di controllo principali e l'Ethernet per le comunicazioni fra gli altri componenti SCADA.

- **Sistemi SCADA IoT**. È l'architettura più attuale, in cui i sistemi SCADA sono integrati con dispositivi IoT e ambienti cloud, portando così a una costante connessione a internet.

L'evoluzione dei sistemi SCADA ha tenuto in considerazione soprattutto l'efficienza, e ben poco la sicurezza. Questo ha avuto come inevitabile effetto la comparsa di nuove minacce, da cui ora più che mai c'è la necessità di difendersi.

3.3 Protocolli tipici dei sistemi SCADA

Per un'analisi completa del funzionamento dei sistemi SCADA, così come per la comprensione degli esperimenti che saranno presentati successivamente, è importante analizzare alcuni dei protocolli tipici più comunemente riscontrabili su reti OT. A dipendenza delle scelte fatte e dei macchinari utilizzati è possibile riscontrare, su reti diverse, diversi tipi di protocolli. Verranno qui analizzati nello specifico due protocolli: Modbus e S7comm. Questi sono infatti i due protocolli presenti nella rete presa in esame negli esperimenti, che verranno analizzati nella sezione 6.

3.3.1 Protocollo Modbus

Le informazioni qui presentate sono state ottenute dalla documentazione riguardante il protocollo Modbus presente sul sito della Modbus Organization [31] [32], così come dalla seguente pagina web della compagnia Control Solutions Inc. [16]

Modbus è un protocollo che si posiziona al livello 7 dello stack ISO/OSI, ovvero il livello applicazione, e si basa sul modello client-server. Tipicamente, ma non necessariamente, la porta su cui le applicazioni che comunicano tramite questo protocollo ascoltano è la 502.
Sono presenti varie versioni del protocollo Modbus, ma le due più importanti sono Modbus RTU, che si basa su comunicazioni seriali, e Modbus TCP/IP, che incapsula richieste e risposte del Modbus RTU in pacchetti TCP.
Un generico frame Modbus, ovvero il suo ADU, che sta per *Application Data Unit*, è visualizzato

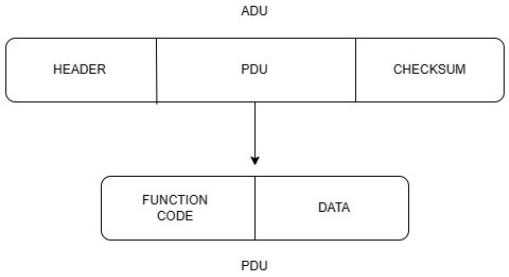

Figura 1: Composizione della ADU per il protocollo Modbus

nell'immagine 1. Questo frame è, nella sua versione base, composto da un header, che può contenere varie informazioni tra cui, tipicamente, gli indirizzi dei dispositivi che comunicano, la PDU, o *Protocol Data Unit*, che contiene i dati della richiesta o risposta, e una checksum in coda. A dipendenza dei protocolli di comunicazione sottostanti, il frame Modbus può essere poi modificato. Per esempio, in Modbus TCP/IP non esiste la checksum finale, dato che il controllo degli errori è gestito dagli strati inferiori dello stack protocollare. L'unica parte che rimane sempre costante è la PDU, che conterrà sempre il codice funzione ed eventuali dati aggiuntivi, come l'indirizzo dell'oggetto da leggere.
Come menzionato, il protocollo Modbus utilizza dei codici funzione per identificare la tipologia di richiesta o risposta che sta venendo trasportata. Il protocollo supporta i codici funzione dall'1 al 255, tuttavia quelli compresi nel range 128-255 sono riservati alla gestione delle eccezioni. La presenza del campo "data" nella PDU dipende dalla funzione.
Modbus modella i dati con cui lavora in quattro tipologie di oggetti:

- Le **Coil**. Dati a 1 bit, modificabili dalle applicazioni.
- I **Discrete Input**. Dati a 1 bit di sola lettura, tipicamente forniti da sistemi di I/O.
- Gli **Holding Register**. Dati a 16 bit, modificabili dalle applicazioni.
- Gli **Input Register**. Dati a 16 bit di sola lettura, tipicamente forniti da sistemi di I/O.

Per ciascuna di queste quattro tipologie di oggetti il protocollo sfrutta uno spazio degli indirizzi a 65536 valori.
Le funzioni di base, quelle più comunemente utilizzate da dispositivi che comunicano tramite il protocollo Modbus, sono visionabili nella tabella 1.

Codice funzione	Funzione
1	Read Coil
2	Read Discrete Input
3	Read Holding Registers
4	Read Input Registers
5	Write Single Coil
6	Write Single Holding Register
15	Write Multiple Coils
16	Write Multiple Holding Registers

Tabella 1: Codici funzione più comuni per il protocollo Modbus

3.3.2 Protocollo S7comm

Contrariamente a quanto visto per Modbus, per S7comm non esiste una documentazione ufficiale. Per questo motivo la seguente analisi si basa sulla documentazione della libreria Snap7 [29], sulla pagina di Wireshark relativa a questo protocollo [40], e sulle informazioni raccolte nel seguente blog [25] [26].

Il protocollo S7comm è un protocollo di comunicazione di tipo client-server progettato per i

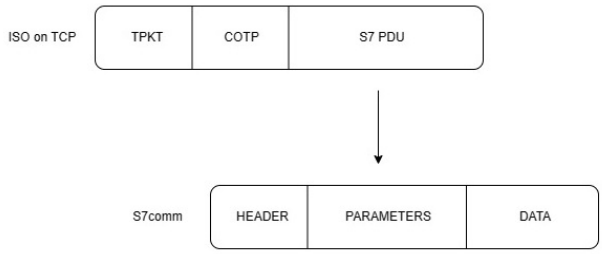

Figura 2: Composizione di un messaggio in S7comm

dispositivi prodotti dall'azienda Siemens. Tipicamente, la porta di ascolto per questo protocollo è la 102. La PDU del protocollo viene trasmessa come payload di un messaggio COTP, che viaggia in TCP/IP grazie al protocollo ISO on TCP. I vari livelli di incapsulamento sono visionabili nell'immagine 2. La PDU è composta da 3 sezioni principali: un header, che contiene informazioni come la lunghezza e il tipo di messaggio, dei parametri, che dipendono dal tipo di funzione, e infine dei dati, anch'essi che dipendono dal tipo di funzione.

Questo protocollo possiede solo due funzioni per la lettura e per la scrittura: con i parametri inseriti nella PDU specifica la quantità di dati da leggere e scrivere e la loro tipologia. I rispettivi codici funzione sono il 4 e il 5. Per indicare l'indirizzo di memoria delle variabili da leggere, oltre che la posizione di partenza (descritta come offset dall'indirizzo di base) e la quantità, va specificata anche l'area di memoria in cui si trovano. La tipologia di area identifica, tipicamente, il tipo di dato. Le principali sono visionabili nella tabella 2. Per esempio, se l'oggetto di una lettura è identificato (basandosi sulla sintassi di Wireshark) da "DB 900.DBX 2.0 BYTE 1", si parla di una

lettura di 1 byte, che si trova a offset di byte 2 e offset di bit 0 nel Data Block 900. Nel dataset costruito per il lavoro qui presentato sono presenti solamente le aree DB.

Sigla area di memoria	Area di memoria	Descrizione
DB	Data Block	Dati generici, l'area più tipicamente utilizzata
M	Merker	Variabili marker o flag
I	Input	Valori di input digitali o analogici
O	Output	Valori di output digitali o analogici
T	Timer	Valore di timer utilizzati dai PLC

Tabella 2: Suddivisione logica delle aree di memoria in S7comm

3.4 Attacchi alle reti OT

Le reti in cui sono presenti i sistemi SCADA possono essere vittime di un grande numero di attacchi informatici, che possono essere sia attacchi comunemente eseguiti anche su reti IT sia attacchi specifici per le reti OT.

3.4.1 Attacchi dall'IT all'OT

Per prima cosa, è importante realizzare quanto le differenze fra le reti IT classiche e quelle OT si siano ridotte. Il che non significa ovviamente che le reti OT non abbiano le loro peculiarità, tutt'altro, ma che aspetti che prima non toccavano minimamente le reti industriali ora lo fanno in maniera preponderante. Questo è dovuto alla sempre maggior presenza di dispositivi connessi a internet in queste reti, che aumentano sicuramente le potenzialità produttive ma creano anche nuove e pericolose superfici di attacco. E dunque, tutti quegli attacchi che un tempo erano esclusivi delle reti IT possono ora colpire anche reti OT. In [1] è presente una estesa dissertazione delle tipologie di attacco che possono colpire le reti SCADA. Gli autori enumerano le varie classi di vulnerabilità che possono trovarsi in queste reti, con varie sotto-categorie. Si parla sia di problematiche di implementazione sia di configurazione. Nel primo caso rientrano vulnerabilità come mancanza di validazione degli input, mancata limitazione della memoria dei buffer, controllo errato del flusso del programma. Tutto ciò può portare a subire attacchi come SQL e command injection, path traversal, abuso di buffer overflow, abuso delle race condition. Nella seconda categoria rientrano invece problematiche come mancanza di cifratura adeguata dei dati, l'utilizzo di password deboli, errori nella configurazione del controllo degli accessi, errori nella gestione del processo di autenticazione.

Come già detto, queste vulnerabilità nascono nel mondo IT, ma con l'introduzione di prodotti *off-the-shelf* e di protocolli tipici dell'IT e dell'integrazione di dispositivi IoT e del cloud, sono comparse anche nel mondo OT. Tutto questo ha anche in generale facilitato gli attacchi nei confronti di sistemi industriali, per via della standardizzazione delle loro componenti. La quantità di informazioni necessarie per impattare seriamente una rete OT è infatti scesa molto.

3.4.2 Attacchi specifici per i sistemi SCADA

Oltre a tutte le vulnerabilità e attacchi che i sistemi OT hanno col tempo ereditato dalle più classiche reti IT, ci sono anche chiaramente attacchi che mirano specificatamente alle reti industriali, che possono poi anche entrare in combinazione con le vulnerabilità analizzate nel paragrafo precedente. L'articolo [27] fornisce una categorizzazione di questi attacchi, specificatamente per il protocollo Modbus (ma tendenzialmente espandibile anche per altri protocolli), dividendoli in quattro gruppi principali. Questi gruppi sono:

- **Attacchi di ricognizione**. Questi sono attacchi che hanno lo scopo di raccogliere informazioni sul sistema in questione, con l'obiettivo di comprenderlo maggiormente per poter determinare la migliore modalità di attacco effettivo. È quindi una sorta di attacco preliminare, che funge da preparazione per quelli futuri. Alcuni esempi di questa tipologia di attacchi possono essere:
  - **Scan degli indirizzi**. Questo attacco ha lo scopo di scoprire quali server sono connessi sulla rete SCADA. È eseguito facendo delle richieste a diversi indirizzi, e controllando per quali di essi si riceve una risposta.
  - **Scan dei *function code***. Questo attacco ha lo scopo di identificare le funzioni che sono supportate dal sistema che si sta analizzando.
  - **Scan dei dispositivi**. Questo attacco identifica le caratteristiche dei singoli dispositivi che fanno parte della rete, come il produttore o la versione del firmware.
  - ***Points scan***. Questa tipologia di attacco ha lo scopo di eseguire una mappatura della memoria dei sistemi.
- ***Response Injection***. I sistemi SCADA generalmente nel loro funzionamento utilizzano tecniche di *polling* per monitorare lo stato delle risorse. Lo scopo di questa categoria di attacchi è quello di andare a modificare le risposte alle richieste che vengono fatte, in modo da fornire informazioni scorrette sullo stato del sistema. Ciò rischia di portare a un funzionamento anomalo e potenzialmente pericoloso. Sono evidenziate due sotto-categorie principali:
  - **NMRI, o *naive malicious response injection***. Questa è la tipologia di *response injection* che viene eseguita quando l'attaccante non ha informazioni precise sul funzionamento del sistema. Per causare danni in questo caso si tende a inserire valori estremi nelle risposte, come numeri enormi o negativi, che possono causare errori nella gestione della risposta.
  - **CMRI, o *complex malicious response injection***. In questo caso l'attaccante ha a disposizione informazioni specifiche sulla macchina target, e conosce i valori tipici presenti nei pacchetti scambiati. Può quindi fare modifiche più precise, inserendo valori falsificati ma realistici in altre situazioni. In questo modo si riesce comunque a fornire informazioni false al sistema, ma con valori meno riconoscibili. Per questo, gli attacchi CMRI tendono a essere molto più complessi da rilevare di quelli NMRI. Con attacchi di

tipo CMRI è anche possibile mascherare alterazioni allo stato del sistema eseguiti con attacchi di tipo *Command Injection*, analizzati in seguito.

- ***Command Injection***. Questa categoria di attacchi ha lo scopo di iniettare falsi comandi di controllo o configurazione, per alterare il comportamento del sistema. Questi sono a loro volta divisibili in:
  - **MSCI, o *malicious state command injection***. Questa tipologia di attacchi va a modificare lo stato del sistema di controllo di processo, in modo tale da spostare il sistema in uno stato critico tramite comandi malevoli spediti ai dispositivi sul campo.
  - **MPCI, o *malicious parameter command injection***. Questa tipologia di attacchi bersagliano i PLC, andandone a modificare i *setpoint*, ovverosia i valori target che il sistema di controllo vuole far produrre al processo fisico.
  - **MFCI, o *malicious function code injection***. Questa sotto-categoria di attacchi sfrutta funzioni standard del protocollo di comunicazione, usate però nel momento sbagliato e in una maniera differente da quella originariamente intesa, il che può avere conseguenze pesanti sul sistema. Ne è un esempio l'iniezione di un comando di restart.
- ***Denial of Service***. Attacchi che hanno lo scopo di bloccare il sistema, rendendolo impossibilitato a svolgere le sue normali funzionalità. Questo ha come conseguenza un rallentamento o un blocco completo del processo produttivo.

4 Stato della ricerca

In questa sezione verrà analizzato lo stato della ricerca riguardante gli IDS, con focus sugli ambienti SCADA. Nello specifico, verranno per prima cosa analizzati alcuni studi condotti nel corso degli anni che avevano come scopo la costruzione di un IDS in ambito OT, con particolare focus su quelli basati sul machine learning, data la loro affinità con i testi qui presentati. Successivamente, verrà analizzata più nel dettaglio la disponibilità di dataset pubblici, già in parte discussa nelle sezioni precedenti.

4.1 Ricerche sugli IDS in ambiente SCADA

Come già menzionato, la ricerca in questo specifico campo è più recente della controparte che si interessa delle reti IT, per tutte le motivazioni analizzate precedentemente: la quantità di lavoro svolto è quindi inevitabilmente più limitata. Ciò detto, l'interesse nei sistemi industriali è sempre crescente data la loro criticità nella società contemporanea.

Nel 2009 Linda et al. [23] presentano un algoritmo di classificazione basato su una rete neurale, a cui è stata applicata una combinazione di due metodi di propagazione dell'errore: la retropropagazione dell'errore e l'algoritmo di Levenberg-Marquardt. Per addestrare questo classificatore hanno anche sviluppato un algoritmo di feature extraction a finestra scorrevole per poter produrre un dataset a partire da un testbed di un sistema a valvole per il controllo dei fluidi. Per estrarre le feature si osserva di volta in volta la finestra di pacchetti, che vengono analizzati assieme per poter produrre feature a livello di rete come, per esempio, il numero di IP differenti presenti nella finestra e il tempo medio fra i pacchetti nella finestra. All'estrazione delle feature è anche seguito un processo di selezione tramite la tecnica dell'*leave one out*.

In un articolo del 2010 [14] Gao et al. presentano un sistema di rilevamento intrusioni sviluppato per rilevare attacchi di tipo response injection. Questo IDS è costruito tramite una rete neurale. Il dataset è stato creato dagli stessi ricercatori, sfruttando un sistema di controllo di una cisterna d'acqua. Secondo il funzionamento nominale della cisterna il livello dell'acqua veniva controllato: se scendeva troppo veniva attivata la pompa per riempire la cisterna, se saliva troppo veniva chiusa. Per questa cisterna erano fissati alcuni valori limite: HH (livello alto di sicurezza), H (il set point alto), L (il set point basso) e LL (il livello basso di sicurezza). Il sistema di controllo della cisterna era tale da far rimanere l'acqua tra i livelli L e H. Se venivano superati i livelli di sicurezza era lanciato un allarme. Su questo sistema i ricercatori, per costruire il dataset, hanno eseguito attacchi di tipo response injection alle richieste di lettura dei livelli dell'acqua con lo scopo di portare il livello dell'acqua oltre i set point od oltre i livelli di sicurezza.

Nel 2016 Keliris et al. [18] presentano un sistema di difesa, basato su machine learning supervisionato, pensato per difendersi da attacchi che definiscono *process aware*, ovvero attacchi che sfruttano la conoscenza del processo fisico per fare danni (come, per esempio, la classe di attacchi CMRI descritta nella sezione 3.4.2). Secondo i ricercatori in questo caso le tipologie di difesa che si possono implementare sono di due tipologie. La prima è quella che si basa sul modello dina-

mico del sistema. Un'anomalia, secondo questo approccio, è un comportamento nel sistema fisico non conforme alle leggi fisiche che lo governano. Questo approccio però richiede una conoscenza del sistema fisico tale da produrre un modello matematico sufficientemente preciso, cosa spesso impossibile. Per questo, dicono, tipicamente è preferito un approccio basato su machine learning. Per eseguire i test i ricercatori hanno costruito un sistema *hardware-in-the-loop* (una parte del sistema simulato, una parte composto da sistemi fisici) che riproduce un sistema di processamento chimico. Il sistema di rilevamento prodotto si basa su una Support Vector Machine, addestrata a riconoscere gli attacchi presentati, che sono divisi a dipendenza del target: attacchi ai sensori, agli attuatori e ai controllori.

In un articolo del 2019 [19] (Khan et al.) viene presentato un IDS "multilivello" pensato per gli ambienti SCADA. Questo IDS viene allenato e testato con il dataset pubblico prodotto da Morris et al. in [28], che è disponibile nel precedentemente citato sito [42]. Su questo dataset viene prima eseguita una feature extraction sfruttando tre diverse tecniche in combinazione: la *Principal Component Analysis*, la *Indipendent Component Analysis* e la *Canonical Correlation Analysis*. L'IDS descritto è definito "multilivello" dato che è composto da due fasi: durante la prima vengono analizzati i singoli pacchetti, nella seconda sono invece analizzati insiemi di pacchetti. Più nello specifico, inizialmente il traffico viene analizzato a livello di singoli pacchetti tramite un Bloom Filter, una struttura dati particolarmente leggera utilizzabile per l'anomaly detection. Se un pacchetto viene riconosciuto come malevolo l'IDS lancia l'allarme e il controllo si interrompe. I pacchetti invece non riconosciuti passano per una seconda fase, che sfrutta il KNN per analizzare le relazioni fra i pacchetti. Se infatti un pacchetto viene rilevato vicino nello spazio delle feature a un sufficientemente alto numero di pacchetti riconosciuti come malevoli, questo viene a sua volta categorizzato come malevolo.

In [4] Anton et al. presentano dei test di rilevamento tramite SVM cost-sensitive e Random Forest eseguiti su una coppia di dataset. Il primo è lo stesso utilizzato nel lavoro precedente, [28], ed è quello incentrato sul protocollo Modbus. Il secondo invece, basato sul protocollo standard OPC-UA [30], è un dataset presentato da Anto et al. in [3]. I ricercatori, per prima cosa, eseguono una feature selection per ridurre la dimensionalità dello spazio dello feature, particolarmente grande in entrambi i dataset. Prima dei test con la random forest è stata eseguita una feature extraction tramite PCA, mentre nel caso dell'SVM è stato eseguito uno *zero mean scaling*.

In [13] Gao et al. introducono un modello di rilevamento intrusioni basato su reti neurali. Questo modello è un *ensemble* di diverse reti: reti feed forward e LSTM. Questo perché, secondo gli autori, le reti ricorsive sono più efficaci nel rilevare attacchi temporalmente correlati fra loro, mentre le più classiche reti feed forward riescono con maggiore successo a rilevare attacchi non temporalmente correlati. L'idea è quindi di combinare queste due reti per costruire un classificatore a due fasi: nella prima il traffico viene analizzato parallelamente da una rete feed forward e da una LSTM, nella seconda gli output dei due classificatori sono passati a una terza rete feed forward che classifica i pacchetti come malevoli o come sicuri, scegliendo la predizione più probabilmente vera tra quelle delle due reti della fase precedente, nel caso ci sia discordanza. I risultati ottenuti dai loro test confermano le ipotesi fatte: la feed forward si comporta meglio con gli attacchi temporalmente scorrelati, la LSTM con quelli temporalmente correlati e infine l'*ensemble* risulta migliore di en-

trambe le reti singole.
In un lavoro del 2020 [34] Phillips et al. hanno testato alcuni algoritmi di classificazione su un dataset creato internamente nel laboratorio di protezione delle infrastrutture critiche dell'università del Mississipi. Dopo aver eseguito una feature selection tramite il test Anova, sono stati testati l'SVM, il decision tree, il KNN e l'algoritmo di clustering K-means. Stando ai loro risultati, tra questi quattro algoritmi quello che ha fornito i risultati migliori è l'SVM, mentre il peggiore, specialmente nel caso multiclasse, è il K-means.
In [37] Rosa et al. presentano un sistema di rilevamento di intrusioni e anomalie che sfrutta le informazioni raccolte e integrate da un SIEM, in modo da poter utilizzare una maggiore quantità di informazioni per la rilevazione rispetto a un IDS che lavora su una singola sezione della rete. Gli autori sottolineano l'importanza in un contesto moderno di sfruttare la maggior quantità di informazioni possibile che si ha a disposizione, che siano informazioni di rete o informazioni operative. La classificazione in sé è poi eseguita da un *ensemble* di algoritmi di machine learning. Degno di nota è inoltre lo sfruttamento di Apache Spark come framework sottostante per la gestione del calcolo distribuito.
Nell'articolo [8], Dehlaghi-Ghadim et al. presentano i loro test di classificazione eseguiti su un dataset da loro creato a partire da un testbed di un processo di riempimento di bottiglie, in cui le varie macchine e sistemi industriali (tubature, valvole, nastri trasportatori, cisterne d'acqua etc.) comunicano fra loro sfruttando Modbus TCP/IP. La loro metodologia di produzione del dataset presuppone che l'attaccante abbia ottenuto accesso alla rete OT, ma che non abbia particolari conoscenze sul suo funzionamento effettivo, il che implica la necessità di condurre anche una fase di ricognizione. Gli attacchi eseguiti dai ricercatori sono stati raggruppati in quattro gruppi: ricognizione, reply, man-in-the-middle e DDoS. Dal traffico raccolto vengono poi estratte le feature per costruire il dataset. Queste feature vanno a caratterizzare un flusso di pacchetti. Gli esperimenti eseguiti sono sia di classificazione binaria sia multiclasse. Gli algoritmi testati sono l'albero decisionale, la random forest e la rete neurale.

4.2 Dataset pubblici

Come già presentato nella sezione 2.3.2, nel campo del rilevamento delle intrusioni, e ancor di più nell'ambito OT, i dataset pubblicamente disponibili per la ricerca sono pochi e spesso vecchi e poco vari nella loro composizione. Questo è principalmente dovuto al fatto che ottenere log di traffico con azioni malevole al suo interno correttamente rilevate ed etichettate è molto difficile. Le aziende, principali obiettivi degli attacchi di cui ci si occupa in questo campo, raramente forniscono informazioni su incidenti informatici che hanno subito, per evitare danni di immagine o anche fuoriuscite di informazioni sensibili. Viene rivelato spesso solo lo stretto indispensabile, che però non è sufficiente per gli scopi di ricerca. Questo fatto lascia ai ricercatori l'onere di creare dei dataset autonomamente, che però è tutt'altro che semplice. Come è infatti spiegato in [22], anche se si ha la capacità di simulare degli attacchi a un sistema SCADA, ci sono grosse difficoltà da affrontare. L'approccio teoricamente più semplice è catturare del traffico normale e iniettarci successivamente del traffico malevolo. Tuttavia è necessario fare molta attenzione al tempismo dei pacchetti, in

quanto il traffico, in una rete industriale, ha dei ritmi molto precisi che vanno rispettati per creare un dataset finale verosimile. L'altra opzione è chiaramente eseguire direttamente degli attacchi o su un sistema SCADA reale, che però richiede l'accesso a hardware anche costoso e non semplice da mantenere, oppure su delle simulazioni, che però, come è sottolineato in [21], hanno spesso problemi di accuratezza.
Un'altra problematica rilevante poi è che i dataset sono costruiti sulla base di una specifica rete, con la sua topologia, i suoi sistemi e i suoi protocolli di comunicazione. Quello dei protocolli è un problema particolarmente grande per la costruzione di NIDS, dato che nell'ambiente industriale ne sono presenti vari, più o meno diffusi. E quindi i dataset pubblici disponibili spesso contengono solamente traffico di un sottoinsieme di questi protocolli di comunicazione, che non necessariamente è perfettamente adattabile a una rete che ne utilizza altri. Per esempio, Modbus, che sembra essere il protocollo più diffuso, è presente nella stragrande maggioranza dei dataset a disposizione. Al contrario, un protocollo come Profinet è quasi assente.
I dataset pubblici più interessanti nel campo del rilevamento di intrusioni in reti industriali sono i seguenti:

- In [42] sono raccolti dei dataset costruiti a partire da reti industriali di diversa tipologia. Da segnalare che i dataset della sezione 3 sono risultati non adatti allo scopo, per via della presenza di pattern non realistici che portavano all'overfitting. Il dataset nella sezione 4 è stato costruito appositamente per contrastare queste problematiche.

- In [35] è presentato un dataset di traffico su una rete di un impianto per raffinazione mineraria. Il protocollo di comunicazione in questo caso è S7Comm.

- Nel sito [9] è presente un dataset costruito a partire da un impianto di sottostazione elettrica ferroviaria. Sono presenti i protocolli Modbus e S7Comm. Questo dataset è presentato e testato nell'articolo [15].

5 Produzione del dataset

In questa sezione verrà presentato il lavoro svolto per la produzione del dataset che verrà poi utilizzato per l'addestramento e test dei vari classificatori. Questi classificatori sono pensati per poter, in futuro, essere sfruttati per la produzione di un IDS che possa osservare il traffico della rete industriale di riferimento e rilevare correttamente del traffico anomalo.
Data la mancanza di dataset pubblicamente disponibili sufficientemente adatti per essere utilizzati nella specifica rete di riferimento, la produzione interna del dataset si è rivelata essenzialmente necessaria. Questo dataset è prodotto a partire dal traffico della rete di riferimento, il quale contiene sia le comunicazioni standard dei dispositivi, sia degli attacchi lanciati da una macchina apposita, in modo da avere a disposizione anche del traffico malevolo, indispensabile per allenare l'IDS. La cattura del traffico è stata eseguita durante il normale funzionamento della rete. Questa caratteristica fondamentale del dataset prodotto fornisce a esso vantaggi e svantaggi che verranno analizzati successivamente.
Verranno ora descritti l'ambito in cui si è lavorato più nello specifico e la modalità con cui si è costruito il dataset, con particolare focus sulla generazione di traffico malevolo

5.1 Descrizione della rete target

La rete industriale utilizzata per la costruzione del dataset necessario per l'allenamento dell'IDS è una rete di cogenerazione per il teleriscaldamento, gentilmente fornita dall'azienda Astea S.p.A. Essa è composta da vari dispositivi connessi in una VLAN, che comunicano con protocolli di tipo diverso, sia protocolli tipici come il TCP, sia protocolli industriali come il Modbus (sia seriale sia TCP/IP), o l'S7comm.
Per catturare il traffico e per eseguire gli attacchi è stato aggiunto un computer alla rete, che simula una macchina compromessa da cui un attaccante può eseguire i propri attacchi. La macchina è stata collegata a due porte diverse dello switch che gestisce il traffico, con quindi due diverse interfacce di rete. Una delle due porte è stata programmata per eseguire il port mirroring di tutte le altre porte dello switch, in modo tale da poter catturare tutto il traffico passante per esso. Tramite l'altra porta (e quindi l'altra interfaccia di rete), il computer può comunicare con il resto dei dispositivi. La cattura è stata eseguita tramite il software Wireshark. La produzione del dataset in sé è descritta nella sezione 5.2.
Nel traffico standard della rete leggibile con questa configurazione sono evidenziabili dieci dispositivi della rete che comunicano, più ulteriori indirizzi IP che non sono di interesse per le analisi qui presentate. Questi dispositivi sono PLC, attuatori e PC di supervisione. A questi si aggiunge, come già detto, il computer utilizzato per gli attacchi, che ottiene così un suo indirizzo IP interno alla VLAN.
Si evidenza anche come nella rete di riferimento siano presenti ulteriori dispositivi il cui traffico non viene indirizzato allo switch utilizzato, e dunque non viene catturato.

5.2 Generazione del dataset

La produzione del dataset ha avuto tre fasi principali: la preparazione degli attacchi da eseguire, l'esecuzione degli attacchi durante la cattura del traffico e la costruzione del dataset effettivo a partire dal file pcapng generato da Wireshark.

5.2.1 Attacchi

Per il lavoro qui descritto ci si è concentrati su due protocolli industriali, il Modbus TCP/IP e l'S7comm. L'approccio utilizzato per entrambi è stato il medesimo: costruire degli attacchi di ricognizione, di lettura e di scrittura da indirizzare a degli specifici dispositivi della rete. La possibilità di lavorare con una rete reale e in normale operatività ha un vantaggio non da poco: permette infatti di ottenere un dataset prodotto da del traffico reale e non simulato. Tuttavia, ciò ha anche portato a una grossa problematica, ovvero il fatto che la tipologia e l'entità degli attacchi eseguibili è limitata dall'ovvia necessità di non causare dei danni al lavoro dell'azienda. Questo è un problema che interessa principalmente le scritture, come verrà analizzato successivamente.
In tutti i casi gli attacchi si basano sul fatto che il traffico nella rete non è né cifrato né autenticato. Questa è una problematica enorme dei protocolli industriali più diffusi, che sono stati sviluppati per l'efficienza e non per la sicurezza. Questo significa che un utente malevolo, se riesce ad avere libero accesso alla VLAN, può fare richieste ai vari dispositivi senza alcuna limitazione, anche senza controllare il dispositivo che di norma dovrebbe eseguire tali richieste.
Per eseguire gli attacchi è stato scritto uno script in Python, il quale seleziona casualmente uno fra gli attacchi possibili, se necessario seleziona casualmente dei parametri e lancia un altro script, relativo a quello specifico attacco, passandogli i parametri necessari o come argomento della chiamata o, nei casi in cui era richiesto dallo script un input dell'utente, passando questi parametri nello standard input. La scelta casuale chiaramente non rappresenta il modo di fare di un attaccante reale, ma poiché le feature verranno estratte dai singoli pacchetti raccolti, non fa differenza.
La prima classe di attacchi è quella che sfrutta il protocollo Modbus. Nel traffico standard sono presenti due macchine che comunicano tramite questo protocollo: un controllore Jace 8000 prodotto dalla Tridium [17], e un controller SIMATIC S7-1200 prodotto dalla Siemens [38]. Il primo fa, durante tutto il traffico catturato, un solo tipo di richiesta al secondo: la lettura di holding register.
Gli attacchi preparati colpiscono il dispositivo della Siemens. Per eseguire gli attacchi si è sfruttato il framework di penetration testing "smod", ottenibile al relativo repository di GitHub [11]. Questo framework, scritto in Python 2, implementa completamente il protocollo Modbus, sfruttando la libreria scapy [41]. smod offre degli script già pronti per essere sfruttati, i quali sono stati utilizzati per lo più così come sono. Per alcuni di essi sono state però necessarie delle modifiche per risolvere alcuni problemi. La maggiore di queste è stata nel file uid.py, che non funzionava correttamente dato che l'eccezione che avrebbe dovuto lanciare in determinate situazioni non veniva mai lanciata. In ogni caso, la struttura e il funzionamento di base di questi script non è stata modificata.

Gli attacchi preparati sono i seguenti:

- **Attacchi di ricognizione**. Gli attacchi sono tre:
  - **Scan del protocollo Modbus**. Questo attacco scansiona vari IP della rete per scoprire quali dispositivi supportano il protocollo Modbus. Per farlo tenta di stabilire una connessione TCP sulla porta di Modbus. Se il 3-way handshake va a buon fine, allora il protocollo è supportato.
  - **Identificazione dell'UID**. Ogni Modbus slave è identificato da un proprio ID. Questo attacco lo identifica con un approccio a forza bruta.
  - **Identificazione delle funzioni supportate**. Questo attacco ha lo scopo di capire quali funzioni sono supportate dal dispositivo target. Per farlo manda delle richieste contenenti le varie funzioni possibili, in alcuni casi malformate per evitare che vengano eseguite e facciano danni già a questo stadio. Se la risposta che si ottiene indica che la funzione viene riconosciuta correttamente, allora la funzione è supportata. Se invece la risposta contiene un'eccezione che indica che la funzione non è supportata, chiaramente ci troviamo nel caso opposto.
- **Attacchi in lettura**. Sono presenti quattro attacchi, uno per ciascun tipo di oggetto Modbus. Lo script seleziona casualmente l'indirizzo da cui partire (nel range di indirizzi leggibili) e la quantità di oggetti da leggere.
- **Attacchi in scrittura**. In questo caso, sono presenti due attacchi:
  - **Scrittura di un *holding register***. Questo attacco scrive un valore casuale tra 0 e 2^{16}-1 sull'*holding register* di indirizzo 0. Essendo il registro 0 un registro di test, è stato possibile eseguire scritture senza correre il pericolo di andare a modificare dati critici per il funzionamento dell'impianto.
  - **Scrittura di coil**. Questo attacco scrive un valore casuale tra 0 e 1 in una delle coil disponibili. In questo specifico caso le coil non sono utilizzate per il funzionamento dei dispositivi, quindi le scritture sono state eseguite senza particolari restrizioni.

Per quanto riguarda invece il protocollo S7comm, anche in questo caso il traffico legittimo presenta una coppia di dispositivi che comunicano tramite questo protocollo. Il primo è un controllore SIMATIC S7-1500 della Siemens [39], mentre il secondo è un modulo periferico SIMATIC ET 200SP [10], anch'esso prodotto dalla Siemens. Il primo fa richieste di lettura e scrittura al secondo in zone diverse della memoria Data Block.
Anche per gli script di attacco per il protocollo S7comm si è sfruttato un framework di penetration testing esistente, scritto in Python 3: ICS-Hacking, anch'esso ottenuto dal relativo repository GitHub [24]. Dei vari protocolli gestiti da questo progetto è stata presa la sezione relativa all'S7comm. Anche ICS-Hacking presenta script di attacco già pronti, ma in questo caso sono stati creati degli script separati, costruiti a partire dagli esempi che il repository contiene. L'eccezione è lo script di scan del protocollo, che invece è uguale a quello Modbus, semplicemente su una porta differente.
Per l'esecuzione degli attacchi in S7comm è stato costruito un Data Block apposito, che contiene

un contatore che autonomamente viene incrementato fino a 1000 per poi essere azzerato, in un loop infinito. È poi presente anche una variabile booleana che, se posta a 1, riporta il contatore a 0. Per richiesta dell'azienda che ha offerto la sua rete per eseguire i test però, questo Data Block non è stato preparato nei due dispositivi menzionati in precedenza, ma su un terzo dispositivo che accetta richieste S7comm ma che normalmente non partecipa al traffico con questo protocollo. Questo dispositivo, scelto come target, è un controllore SIMATIC S71200 (differente da quello usato come target per gli attacchi Modbus).
Gli attacchi presenti sono quindi:

- **Scan del protocollo S7comm** come attacco di ricognizione. Così come nel caso di Modbus, si tenta l'instaurazione di una connessione TCP sulla porta relativa a questo protocollo. Se la cosa va a buon fine, il protocollo è supportato.
- **Attacco in lettura**. Questo attacco legge il valore del contatore.
- **Attacco in scrittura**. Questo attacco esegue una scrittura del valore 1 sulla variabile che resetta il contatore.

5.2.2 Generazione del traffico

Completata la preparazione degli script di attacco, la fase successiva è stata quella di produzione del traffico, da poter poi manipolare per costruire il dataset. Sono stati catturati circa 320.000 pacchetti nel corso del normale funzionamento della rete, durante il quale gli attacchi erano costantemente in esecuzione. Come anticipato, lo script, pensato per essere eseguito in loop, seleziona casualmente un attacco e lo esegue. Tuttavia, un approccio completamente casuale portava a due problemi. Per prima cosa, la distribuzione degli attacchi sarebbe stata incredibilmente sbilanciata nei confronti degli attacchi di scan. Infatti, un singolo attacco di ricognizione produce varie decine di pacchetti che verranno poi classificati come malevoli, mentre gli attacchi di lettura e scrittura ne generano molti di meno singolarmente. Un altro problema poi è la differenza in velocità dei vari attacchi: lo scan dell'S7comm e soprattutto lo scan delle funzioni Modbus sono sensibilmente più lenti degli altri attacchi, il che contribuisce all'esagerato sbilanciamento.
Per mitigare queste problematiche si è fatto in modo che lo scan dell'UID e gli attacchi di lettura e scrittura abbiano una probabilità molto più alta di essere selezionati. Nello specifico, il metodo che si è utilizzato è il seguente:

1. Assegnazione di un identificatore numerico a ogni attacco
2. Costruzione di un array in cui gli id relativi agli scan dei protocolli e delle funzioni Modbus compaiono una sola volta, mentre gli altri id compaiono 50 volte ciascuno.
3. Selezione casuale di uno degli elementi dell'array, che rappresenta l'attacco da far partire.

L'algoritmo di selezione dell'attacco è presentato sotto forma di pseudocodice nell'algoritmo 1.

Algorithm 1 Algoritmo di selezione degli attacchi

INIZIO

Definizione dell'array per la selezione dell'attacco

$attackIdsArray \leftarrow$ Array vuoto

$attackIdsArray[0] \leftarrow 0$

$attackIdsArray[1] \leftarrow 1$

$attackIdsArray[2] \leftarrow 2$

$k \leftarrow 3$

for $i \leftarrow 3, 11$ **do**

  for $j \leftarrow 0, 49$ **do**

    $attackIdsArray[k] \leftarrow i$

    $k{+}{+}$

  end for

end for

Selezione dell'indice ed esecuzione dell'attacco

$x \leftarrow$ selezioneCasuale$(1, \text{len}(attackIdsArray))$

$idScelto \leftarrow attackIdsArray[x]$

if $x = 0$ **then**

  modbusFunctionCodeScan

else if $x = 1$ **then**

  s7commIpScan

else if $x = 2$ **then**

  modbusIpScan

else if $x = 3$ **then**

  modbusUidScan

else if $x = 4$ **then**

  modbusReadCoil

  ...

else if $x = 11$ **then**

  s7commWriteDB

end if

FINE

5.2.3 Estrazione delle feature ed etichettatura

Come precedentemente menzionato, si è deciso di estrarre le feature a partire dai singoli pacchetti, per poi testare le capacità degli algoritmi nel lavorare in questa situazione. Essenzialmente, la tipologia di feature estratte si basa fortemente su quelle presenti in uno dei dataset pubblici presentati nella sezione 4.2, ovvero Electra.
Per produrre il dataset dal traffico raccolto è stato scritto uno script Python che esegue l'estrazione delle feature e l'etichettatura. Per prima cosa il programma carica il file pcapng generato nel passo precedente, con l'ausilio della libreria pyshark [20]. Vengono caricati solamente i pacchetti che viaggiano in TCP: nessun attacco sfrutta protocolli che non si basano sul TCP per la comunicazione, e quindi non sono rilevanti. L'ipotetico IDS potrà poi automaticamente etichettare pacchetti che non viaggiano in TCP come normali. Per ciascun pacchetto del dataset dunque si estraggono le feature di interesse, e poi si etichetta il pacchetto sulla base delle feature estratte. Le feature, così come le etichette, vengono rappresentate con numeri interi già fin da ora, dato che poi sarebbe necessario farlo in ogni caso per proseguire con la classificazione. Dato che non in tutti i pacchetti sono presenti gli stessi dati, va affrontato il problema dei dati mancanti. Per evitare di doverlo gestire successivamente, fin da subito è associato un valore di default nel caso in cui una particolare feature non sia estraibile.
Si riportano nella tabella 3 le feature estratte con il valore di default, se previsto, e la sua modalità di rappresentazione intera.
Una volta estratte le feature, si passa all'etichettatura. Vengono assegnate due etichette:

- Un'etichetta binaria, che distingue fra traffico sicuro e malevolo
- Un'etichetta multiclasse, che distingue fra i singoli attacchi descritti nella sezione 5.2.1

Il dataset così costruito ha ancora un grande problema di sbilanciamento: il traffico sicuro è estremamente prevalente. Le tabelle 4 e 5 mostrano le proporzioni in relazione alle due diverse etichette. Questo è con alte probabilità un problema, dato che dei classificatori allenati con dataset sbilanciati tipicamente tendono a favorire la classe prevalente durante le predizioni. D'altra parte, è inevitabile che un dataset prodotto in questa modalità presenti un forte sbilanciamento a favore del traffico standard della rete, ed è quindi una questione da affrontare molto spesso in questo campo di studi. Si anticipa che per ridurre gli effetti dello sbilanciamento sui risultati è stata applicata la tecnica dell'undersampling, come verrà descritto nella sezione 6.

Feature	Descrizione
Protocollo	Il protocollo di comunicazione. È stato costruito un dizionario che associa ciascun protocollo presente con un numero intero.
IP sorgente	L'indirizzo IP sorgente. La rappresentazione intera viene prodotta eliminando i punti e scrivendo ogni sezione dell'indirizzo in tre cifre, se necessario aggiungendo zeri a sinistra.
IP destinazione	L'indirizzo IP di destinazione. La rappresentazione intera è analoga a quella dell'IP sorgente.
Codice funzione	Il codice funzione. Quello Modbus viene fornito già intero da pyshark, mentre per l'S7comm è in esadecimale, e va quindi convertito. Se assente dal pacchetto, il valore di default è -1.
Registro di partenza	Il registro da cui far partire la lettura/scrittura. È di base un intero. Il valore di default è -1
Quantità di byte	La quantità di byte da leggere o scrivere. Il valore di default è 0.
Quantità di bit	La quantità di bit da leggere o scrivere. Il valore di default è 0.
Data	I dati da scrivere per le operazioni di scrittura. Quelli Modbus sono espressi da pyshark come una sequenza di byte in esadecimale separati dal simbolo ":". Viene rimosso il separatore e l'esadecimale unificato viene convertito in intero. Il relativo campo per S7comm è invece già un intero. Il valore di default è -1.

Tabella 3: Feature presenti nel dataset

Label	Quantità
Safe	317091
Malicious	8332

Tabella 4: Proporzioni delle classi nel dataset, etichette binarie

Label	Quantità
Safe	317091
Protocol scan	1050
Function scan	1397
UID scan	673
Read Coils	676
Read Discrete Inputs	684
Read Holding Registers	628
Read Input Registers	654
Write Holding Register	632
Write Coil	626
Read DB	649
Write DB	663

Tabella 5: Proporzioni delle classi nel dataset, etichette multiclasse

6 Test di classificazione

Completata la produzione del dataset si è passati ai test di classificazione. Per prima cosa, sono stati testati dei classici algoritmi di classificazione in due task differenti: classificazione binaria e multiclasse.

Successivamente, sono stati anche testati dei classificatori one-class. Rispetto alla grande famiglia di classificatori precedenti, quelli one-class hanno la particolarità di essere allenati, come si intuisce dal nome, tramite un dataset composto esclusivamente da campioni appartenente a una singola classe. Il principio di funzionamento di tali classificatori consiste nello sfruttamento degli elementi del dataset per identificare criteri e caratteristiche funzionali a descrivere la classe di riferimento. Sulla base di quanto appreso, un classificatore one-class stabilirà se i campioni di test che vengono analizzati siano conformi oppure no a quanto osservato nella fase di addestramento. I campioni non conformi ai criteri individuati saranno classificati come anomali. Nel caso specifico preso in esame, la classe di riferimento è quella del traffico sicuro, mentre la classe "anomala" è quella del traffico malevolo. Va da sé che questa categoria di classificatori può eseguire esclusivamente classificazione binaria.

Per valutare le performance dei classificatori sono stati utilizzati tre strumenti:

- Metriche di classificazione generali. Nello specifico, accuratezza, precisione, recall e F1-Score.
- Una 5-fold cross validation, per valutare la stabilità dei risultati. La metrica utilizzata per valutare la cross-validation è l'F1-Score. La cross-validation calcola l'F1-score durante l'allenamento dei modelli. Per questo motivo, non è adatta per i classificatori one-class, che in fase di allenamento vedono solamente una classe. Dunque, è stata utilizzata esclusivamente per i classificatori supervisionati.
- Le matrici di confusione, per visualizzare i risultati per le singole classi.

Vale la pena definire formalmente le quattro metriche di classificazione. Indicando i veri positivi, falsi positivi, veri negativi e falsi negativi rispettivamente con TP, FP, TN e FN, le misure sono definite come segue:

$$\text{Accuratezza} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precisione} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precisione} \cdot \text{Recall}}{\text{Precisione} + \text{Recall}}$$

Queste metriche, così come sono definite, sono pensate per valutare complessivamente solamente una classificazione binaria. Inoltre, a esclusione dell'accuratezza, si riferiscono esclusivamente, nel caso preso in esame, alla classe del traffico malevolo. Per avere delle metriche applicabili anche nel caso multiclasse e che si riferiscano a tutte le classi prese in esame sono applicati due metodi, uno per l'accuratezza e un altro per le altre metriche. Nel caso dell'accuratezza, per la classificazione

binaria non è necessario fare nulla, mentre per quella multiclasse è sufficiente calcolare il rapporto fra predizioni corrette e il numero di predizioni totali. Le altre metriche invece vengono calcolate singolarmente per ogni classe, e poi ne vengono calcolate le medie (si parla di *macro averaging*). In questo modo si ottengono dei singoli valori che indicano la performance generale su tutte le classi.

6.1 Descrizione dei classificatori utilizzati

Prima di passare ai test, viene data una breve descrizione di tutti i classificatori che verranno utilizzati negli esperimenti.

6.1.1 Classificatori supervisionati

I classificatori supervisionati presi in esame, le cui implementazioni sono fornite dalla libreria scikit-learn ([33], [6]), sono l'SVC (*Support Vector Classifier*, comunemente chiamato anche *Support Vector Machine*, o SVM), il KNN (ovvero *K-Nearest Neighbours*, il Naive-Bayes Gaussiano, l'Albero decisionale, l'MLP, la Random Forest, l'Adaboost e il Gradient Boosting. Segue una descrizione più dettagliata di questi algoritmi. Dove specificato, le immagini utilizzate provengono dalla documentazione di scikit-learn, e sono ricreabili sfruttando le API che mettono a disposizione.
La **Support Vector Machine** è un algoritmo di classificazione binaria supervisionato che ha lo scopo di suddividere lo spazio N-dimensionale delle feature in due regioni che isolano il meglio possibile le due classi. Esso cerca di posizionare il bordo di classificazione che divide le due regioni in modo tale che sia il più possibile equamente distante da entrambe le classi. Se le due classi sono linearmente separabili, allora è possibile definire un bordo di classificazione lineare. In caso contrario, viene applicata una tecnica definita *kernel trick*. Viene sfruttata una particolare funzione, detta funzione kernel, per trasportare il problema di classificazione dallo spazio delle feature originale a uno spazio delle feature di dimensione più alta, nel quale gli elementi risultano linearmente separabili. Un esempio visivo di problema bidimensionale linearmente separabile è osservabile nell'immagine 3. Questo algoritmo è espandibile anche per problemi di classificazione a

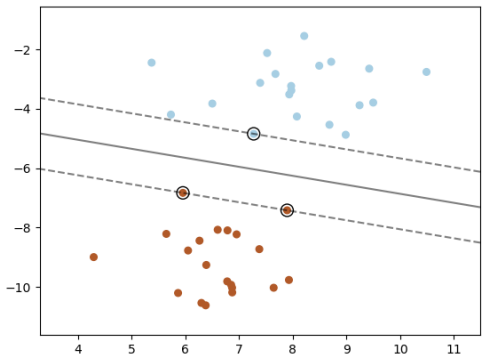

Figura 3: Esempio applicazione SVM [33]

N classi sfruttando uno fra due approcci: il One-Versus-All o il One-Versus-One. Nel primo caso, si esegue una classificazione binaria tra una specifica classe di riferimento e una classe composta

dagli elementi appartenenti a tutte le altre classi. Questa classificazione è ripetuta N volte, una per ogni possibile classe di riferimento. Dall'aggregazione dei risultati prodotti dagli N classificatori si estrae la classe predetta per ciascuna istanza. L'approccio One-Versus-One invece si basa sull'allenamento di una serie di classificatori binari, ciascuno dei quali ha il compito di distinguere fra due delle N classi presenti nel dataset. Vengono così costruiti $\binom{N}{2} = \frac{N!}{2!\,(N-2)!}$ classificatori, i cui risultati vengono combinati per stabilire la classe di appartenenza più probabile del campione. Il **KNN**, sigla che sta per K-Nearest Neighbor, è un algoritmo di classificazione che basa la predizione della classe di appartenenza di ciascun campione sui K campioni a esso più vicini. Nella sua versione più semplice si raccolgono le classi di questi campioni vicini, e la classe che compare in maggioranza è quella che viene selezionata come classe di appartenenza del campione (*majority voting*). L'idea di fondo alla base di questo approccio è che nello spazio delle feature le classi non si dispongono a caso tipicamente, ma in regioni diverse dello spazio. Perciò elementi della stessa classe saranno tipicamente vicini ad altri elementi della stessa classe. Esistono versioni più complesse: per esempio è possibile assegnare un peso diverso alla votazione sulla base della distanza del vicino dal campione studiato. Un esempio di applicazione del KNN è visionabile nell'immagine 4.

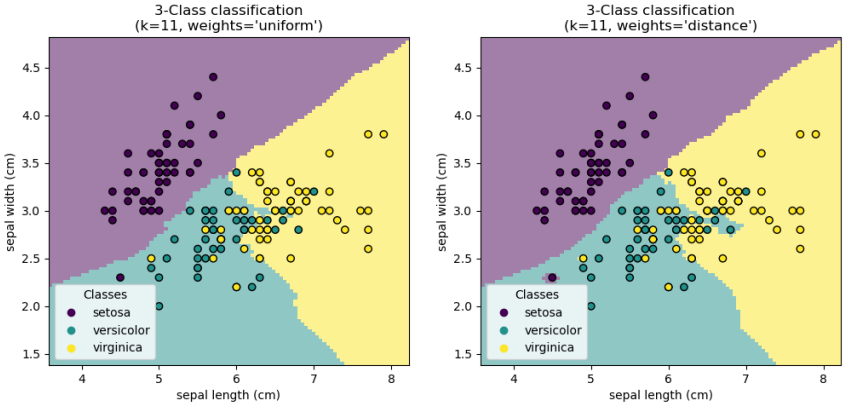

Figura 4: Esempio applicazione KNN [33]

La famiglia di classificatori **Naive Bayes** sono un insieme di classificatori supervisionati che si basano sull'applicazione del teorema di Bayes per il processo decisionale. Detti $x = (x_1, ..., x_n)$ il vettore delle feature e y la variabile di classe, il teorema di Bayes dice che:

$$P(y \mid x) = \frac{P(y)P(x \mid y)}{P(x)} \tag{1}$$

Questi classificatori si basano sull'ipotesi secondo cui $P(x_i \mid y, x1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i \mid y)$. Assunta vera questa ipotesi infatti si può riscrivere la formula 1:

$$P(y \mid x) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(x)} \tag{2}$$

Dato che, preso l'insieme dei campioni, la probabilità *P(x)* è costante, si conclude che la classe predetta sarà:

$$y_p = \underset{y}{\operatorname{argmax}}\, P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{3}$$

Le probabilità $P(y)$ e $P(x_i \mid y)$ vanno calcolate a priori, dove possibile. Molto spesso questo è impossibile, e bisogna stimarle. Le varie tipologie di classificatori Naive Bayes si differenziano sulla base della stima di questa seconda probabilità. Nel caso del Naive Bayes Gaussiano, quello utilizzato in questo caso, si suppone che la distribuzione dei campioni sia una distribuzione Gaussiana, e perciò si stima la probabilità condizionata come $P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$, dove μ_y e σ_y sono rispettivamente media e deviazione standard.

L'**albero decisionale** è un algoritmo di classificazione supervisionato che produce come modello di classificazione una struttura ad albero, in cui ogni nodo corrisponde a una specifica condizione di split, ovvero una condizione algebrica definita su una delle feature che suddivide il dataset in due sottoinsiemi. Ogni split ha lo scopo di incrementare l'omogeneità dell'insieme dei campioni considerato in ciascun nodo dell'albero, fino ad arrivare, idealmente, a degli insiemi perfettamente omogenei, ovvero che contengono solo elementi di una singola classe. A questo punto le predizioni vengono eseguite percorrendo l'albero sulla base delle condizioni di split che vengono verificate o meno. Il percorso nell'albero si concluderà su uno specifico nodo foglia, il quale corrisponde a una delle classi. Il campione sarà quindi classificato sulla base del nodo foglia raggiunto. Un semplice esempio di albero decisionale, supponendo uno spazio tridimensionale e tre possibili classi, è visionabile nell'immagine 5.

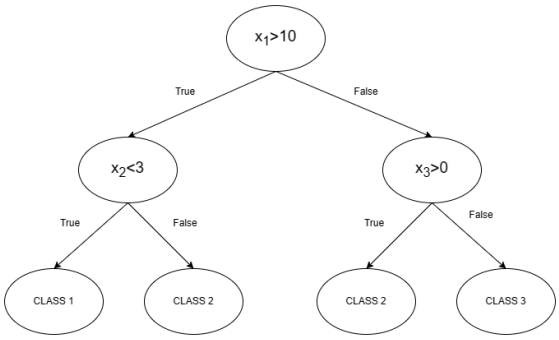

Figura 5: Esempio di Albero Decisionale

L'**MLP**, o *Multi-Layered Perceptron*, è una rete neurale *feed forward*. Una rete neurale è una struttura matematica composta da nodi, o neuroni, collegati fra loro. I nodi sono raggruppati in layer. Le reti neurali più semplici hanno solo due layer: il primo è quello di input, che contiene i dati in ingresso, mentre il secondo è quello il cui output è il vero e proprio output della rete. Reti più complesse (quasi tutte nelle applicazioni reali) hanno anche altri layer intermedi, detti layer nascosti, il cui output è passato al layer successivo. Nelle reti feed-forward i collegamenti vanno tutti in un'unica direzione: dall'input all'output. Una semplice rappresentazione di una rete *feed forward* è rappresentata dall'immagine 6. Ogni input di un nodo è una combinazione lineare di alcuni degli output del layer precedente. I coefficienti delle varie combinazioni lineari sono detti i pesi della rete. Questa combinazione lineare passa poi per una funzione di attivazione che trasla il valore nell'intervallo [0,1]. Funzioni tipiche utilizzate come funzioni di attivazione sono la sigmoide o la arcotangente. Spesso alla combinazione lineare è anche aggiunto un valore detto bias. Formalmente, se si indicano con in_j l'input del nodo j, con $w_{i,j}$ il peso relativo al collegamento fra

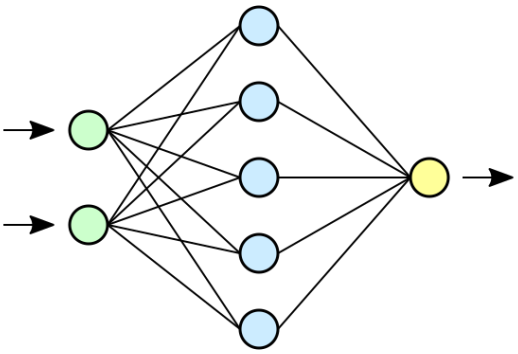

Figura 6: Rappresentazione rete neurale *feed forward* [7]

il nodo i del layer precedente e il nodo j, e con x_i l'output del nodo i, allora possiamo scrivere:

$$y_j = g(w_{0,j} + \sum_{i=1}^{n} w_{i,j} x_i) \tag{4}$$

Allenare una rete neurale significa trovare i valori dei pesi e dei bias che portano a minimizzare l'errore di classificazione.

I classificatori rimanenti sono classificatori di tipo "*ensemble*". Questa tipologia di modelli sono ottenuti dall'unione di più classificatori elementari, la cui combinazione mira a ottenere complessivamente prestazioni migliori rispetto a quelle che si otterrebbero dai singoli. Come anticipato, i tre modelli ad *ensemble* sfruttati sono la Random Forest, l'Adaboost e il Gradient Boosting.

La **Random Forest** sfrutta come classificatori elementari degli alberi decisionali, ognuno dei quali è allenato su un sottoinsieme casuale del training set. Questo approccio è anche detto *bagging*. La predizione complessiva della Random Forest è scelta tramite un sistema di voto a maggioranza.

L'**Adaboost** si basa sulla composizione di un insieme di classificatori che sono definiti "deboli" (detti "*weak learners*"), ovvero solo leggermente migliori di una predizione casuale, allenati con dataset di training diversi. Tipicamente si tratta di piccoli alberi decisionali. La predizione finale si ottiene con un voto a maggioranza pesato dei *weak learners*. Ciò che lo distingue dalla Random Forest è il processo di allenamento, il quale è costituito da una serie di round, in ognuno dei quali vengono modificati dei pesi assegnati agli input. All'inizio questi pesi sono posti a $\frac{1}{N}$, dove N è il numero di campioni, e a ogni round sono aumentati se questi erano stati precedentemente classificati male, o abbassati nel caso opposto. Questo ha lo scopo di far concentrare i *weak learners* sui campioni in cui c'è maggiore difficoltà.

Infine, il **Gradient Boosting** è considerato una generalizzazione dell'Adaboost. Anche in questo caso si sfrutta un allenamento iterativo di un insieme di *weak learners* (alberi decisionali), ma stavolta l'aggiornamento dei parametri è più complesso. Viene infatti introdotta una *loss function* che viene calcolata a ogni iterazione. I pesi vengono quindi aggiornati sulla base del gradiente della *loss function*, in un procedimento idealmente analogo a quello sfruttato per l'aggiornamento dei pesi durante l'allenamento delle reti neurali.

6.1.2 Classificatori One-Class

I modelli appartenenti alla famiglia dei classificatori "*one-class*" hanno la particolarità, come si intuisce dal nome, che per l'allenamento necessitano esclusivamente di campioni di una singola classe, tramite i quali imparano a distinguerla da tutto il resto della popolazione. Nel caso preso in considerazione, quello degli IDS, la classe di allenamento è quella del traffico sicuro, che quindi i classificatori *one-class* imparano a distinguere dal traffico malevolo.

Questa tipologia di classificatori ha chiaramente degli svantaggi non indifferenti: primo fra tutti l'inevitabile incapacità di eseguire classificazione multiclasse, dato che, visto il processo di allenamento, possono solamente imparare a riconoscere un campione come appartenente alla classe di allenamento o non appartenente alla classe di allenamento. Inoltre, tipicamente sono meno performanti dei classificatori ad apprendimento supervisionato più "standard" visti in precedenza. Gli *one-class* hanno però un vantaggio fondamentale: il richiedere solamente una classe per l'allenamento. In questo campo di applicazione questo significa che per allenare un classificatore di questa tipologia non si ha bisogno di raccogliere traffico malevolo, che è la grossa problematica del lavoro nel campo degli IDS. È sufficiente raccogliere del traffico sicuro della propria rete per costruire il dataset di allenamento. Chiaramente, se c'è la necessità di testare il classificatore il traffico malevolo è indispensabile, ma se questo passaggio non interessa (magari perché il modello che si sta allenando è già stato testato da altri in precedenza), allora il processo di allenamento è molto facilitato.

I classificatori testati per questi esperimenti sono il One Class SVM, il Feature Bagging, la Histogram-based Outlier Detection (HBOS), l'Isolation Forest, il One Class KNN e una versione del One Class KNN detto Average KNN, che invece di sfruttare la distanza dal k-esimo elemento più vicino sfrutta la media delle distanze di tutti i k-esimi elementi più vicini. Segue una descrizione per ciascuno di questi classificatori.

Il **One Class SVM** è una versione particolare della Support Vector Machine, presentata nella sezione 6.1.1, pensata per essere allenata con elementi di una singola classe. In questo caso l'idea è quella di definire, nello spazio delle feature, un bordo decisionale capace di contenere tutti gli elementi della classe presa in considerazione, costruito in modo tale da minimizzare la distanza tra di esso e gli elementi. Nell'immagine 7 è mostrato il One Class SVM applicato al popolare dataset Iris [12], considerate solo le prime due feature per avere un grafico bidimensionale.

Il **Feature Bagging** è un algoritmo ad *ensemble* che combina i risultati di più algoritmi di rilevamento di anomalie, ciascuno dei quali utilizza un piccolo sottoinsieme di feature selezionate in modo casuale dal set di feature originale. Di conseguenza, ogni rilevatore identifica anomalie diverse. A ogni istanza del dataset vengono assegnati dal processo di classificazione dei punteggi che corrispondono alla loro probabilità di essere anomali. I punteggi calcolati dai singoli algoritmi di rilevamento vengono combinati per trovare, idealmente, tutti gli outlier. I *weak learner* utilizzati nell'*ensemble* possono essere prodotti sfruttando modelli diversi. Nel caso degli esperimenti descritti successivamente, è stato utilizzato il **LOF**, o *Local Outlier Factor*, un algoritmo di anomaly detection non supervisionato che basa il suo funzionamento sul concetto di *densità locale*, secondo cui la località è data dai k elementi più vicini. Nello specifico, la distanza da questi k elementi più

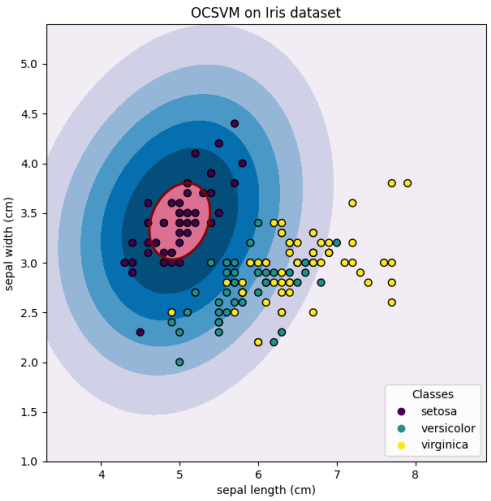

Figura 7: Applicazione OCSVM su Iris

vicini viene usata per stimare la densità della regione di spazio attorno all'elemento considerato. Dal confronto tra la densità locale di un campione rispetto a quella dei suoi vicini, si identificano regioni di densità simile e regioni che hanno una densità inferiore. Elementi che si trovano in queste seconde regioni saranno considerati come elementi anomali. Un esempio di applicazione del LOF è presente nell'immagine 8.

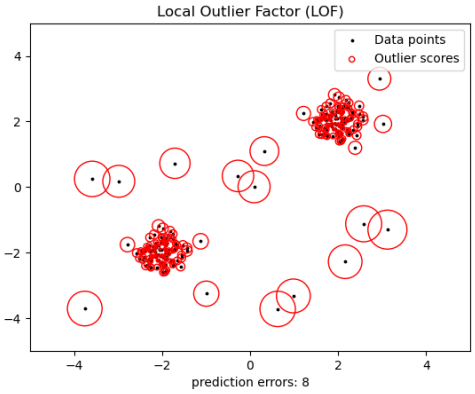

Figura 8: Applicazione LOF [33]

La **Histogram-based Outlier Detection**, o HBOS, è un modello di classificazione che basa il suo funzionamento su un'analisi univariata. Uno dei suoi punti di forza maggiori è l'essere in grado di eseguire predizioni sulle istanze di dati analizzate mantenendo basso il costo computazionale. Il suo funzionamento si basa sulla produzione di un istogramma per ciascuna feature del dataset. Se la feature è di tipo categorico, sono utilizzate le frequenze dei possibili valori di quella feature, mentre nel caso di feature numeriche ciascun *bin* dell'istogramma indica un range di valori possibili. Questi istogrammi sono poi normalizzati tra 0 e 1. Per eseguire la predizione viene quindi calcolato un *anomaly score*. Indicando con n il numero di feature, il punteggio dell'elemento x è

definito in questo modo:

$$HBOS(x) = \sum_{i=0}^{n} log(\frac{1}{hist_i(x)})$$

Minore è il punteggio ottenuto, e quindi minore il valore dei bin di ciascuna feature in cui x ricade, maggiore è la probabilità che quell'istanza sia anomala.

La **Isolation Forest** è un algoritmo di rilevamento anomalie ad *ensemble* che sfrutta alberi decisionali come *weak learner* (in maniera analoga alla Random Forest) allenati tramite dataset sempre diversi, tipicamente realizzati come sottoinsiemi casuali del dataset completo. L'ipotesi alla base di questo modello è che risulta più facile isolare dei campioni anomali rispetto a campioni ordinari. Dunque, gli alberi decisionali richiederanno un numero minore di split per arrivare al nodo foglia che rappresenta un elemento anomalo. In altre parole, il percorso dalla radice dell'albero alla foglia in cui si trova l'anomalia sarà più breve rispetto al percorso necessario per raggiungere le osservazioni normali. Questo concetto viene utilizzato per la definizione di un *anomaly score* che valuta per ciascuna istanza del dataset la lunghezza media del percorso per ciascuno degli alberi casuali creati. Più brevi saranno i percorsi per un'istanza nei vari alberi, maggiore sarà il punteggio ottenuto e quindi maggiore sarà la probabilità che l'istanza valutata sia un'anomalia.

Il KNN è già stato descritto nella sezione 6.1.1, tuttavia in questo caso si parla di una versione diversa che, per l'appunto, sfrutta solamente una classe in allenamento. L'idea alla base è in ogni caso analoga: si utilizzano le distanze dai K elementi più vicini. Sono state sfruttate, come preannunciato, due versioni del **One Class KNN**. Quella base utilizza la distanza dal K-esimo elemento più vicino, mentre la versione "*average*" calcola la media delle distanze da tutti i K elementi più vicini. Questo valore, sia nel primo sia nel secondo caso, viene utilizzato come *outlier score*.

6.2 Esperimenti con i classificatori supervisionati

Come anticipato, per lavorare con questi classificatori si è fatto uso della libreria scikit-learn. Si specifica inoltre che i classificatori sono stati utilizzati senza toccare i parametri di default, esclusa la rete MLP, per cui è stata specificata una dimensione del batch di 105. Per ogni classificatore con delle componenti di casualità è stato fissato il seed del PNRG, per garantire il mantenimento degli stessi risultati.

Per ciascuna delle due tipologie di classificazione, i test sono ripetuti con e senza sotto-campionamento del dataset, utilizzato per tentare di ridurre gli effetti dello sbilanciamento sui risultati. Sono stati poi eseguiti ulteriori test eliminando le informazioni di provenienza e destinazione dei pacchetti dalle feature, ovvero gli indirizzi IP e la porta di destinazione. Questo è stato fatto per testare la capacità dei modelli di distinguere correttamente fra le varie classi sfruttando esclusivamente il protocollo utilizzato e il contenuto del pacchetto, senza tenere in considerazione da dove proviene o dove è diretto. Il motivo principale per cui queste feature potrebbero voler essere tolte è che utilizzandole il classificatore rischia di basarsi eccessivamente su di esse, cadendo in overfitting. Se, per esempio, il classificatore imparasse a riconoscere come malevolo del traffico proveniente dallo specifico indirizzo IP da cui sono partiti gli attacchi durante la generazione del traffico, successivamente, in presenza di attacchi provenienti da un diverso indirizzo, il classificatore potrebbe non essere in grado di riconoscerli. Per eseguire questi esperimenti sono quindi stati costruiti, a partire

dal dataset originale prodotto secondo le modalità descritte nel paragrafo 5.2, quattro dataset diversi, sia per il caso binario sia per quello multiclasse. Vengono denominati secondo la tabella 6.

Dataset	Undersampling	Rimozione feature
D1	NO	NO
D2	SÌ	NO
D3	NO	SÌ
D4	SÌ	SÌ

Tabella 6: Dataset per i vari esperimenti

Ciascuno dei quattro dataset passa per una fase di preprocessamento, che viene qui descritta più nel dettaglio. Il pre-processamento eseguito è, tranne dove specificato, identico nel caso binario e multiclasse. I pre-processamenti eseguiti sono i seguenti:

- **Dataset D1**. Viene per prima cosa viene eseguito un hold-out del 20% sul dataset, per eseguire il test. Questo split è uno split stratificato sulle classi di attacco, con lo scopo di mantenere le loro proporzioni sia sul training set sia sul test set. Dopo aver eseguito lo split, le feature del train e del test set sono scalate tra 0 e 1. Questo è indispensabile per alcune delle tipologie di classificatori utilizzati, come l'SVM, dato che nel dataset utilizzato le scale numeriche delle varie feature sono molto differenti. Questo è particolarmente vero per gli indirizzi IP, che sono numeri a 12 cifre.

- **Dataset D2**. Rispetto al dataset D1, lo split è un po' più complesso, ed è leggermente diverso nel caso binario e nel caso multiclasse. Nel primo caso, viene eseguito uno split 80-20 sulla classe del traffico malevolo, il quale non ha necessità di essere sotto-campionato. Nel caso multiclasse invece lo split 80-20 è eseguito singolarmente su ciascuna classe di attacco. In entrambi i casi, successivamente, vengono presi un numero campioni della classe di traffico sicuro tali da averne tanti quanti sono quelli malevoli. Tutti gli altri campioni finiscono nel test set, in modo tale che tutto il dataset venga utilizzato in qualche modo. La scalatura è sempre la medesima del dataset D1.

- **Dataset D3**. Come il dataset D1, ma viene inizialmente eseguita la rimozione delle feature che contengono gli indirizzi IP e la porta di destinazione.

- **Dataset D4**. Come per il dataset D2, ma viene inizialmente eseguita la rimozione delle feature che contengono gli indirizzi IP e la porta di destinazione.

Va sottolineato in oltre come le operazioni di split hanno un seed del PNRG fissato, in modo tale da ottenere gli stessi risultati con esecuzioni ripetute. Le proporzioni delle classi presenti nei dataset di training e test sono riportate nelle tabelle 7, 8, 9 e 10.

Dataset	Safe	Malicious
Training set	253673	6665
Test set	63418	1667

Tabella 7: Composizione di training set e test set, dataset D1 e D3, classificazione binaria

Dataset	Safe	Malicious
Training set	6666	6666
Test set	310425	1666

Tabella 8: Composizione di training set e test set, dataset D2 e D4, classificazione binaria

Classe	Training set	Test set
Safe	253673	63418
Protocol scan	840	210
Function scan	1118	279
UID scan	538	135
Read Coils	541	135
Read Discrete Inputs	547	137
Read Holding Registers	502	126
Read Input Registers	523	131
Write Holding Register	506	126
Write Coil	501	125
Read DB	519	130
Write DB	530	133

Tabella 9: Composizione di training set e test set, dataset D1 e D3, classificazione multiclasse

Classe	Training set	Test set
Safe	6665	310426
Protocol scan	840	210
Function scan	1118	279
UID scan	538	135
Read Coils	541	135
Read Discrete Inputs	547	137
Read Holding Registers	502	126
Read Input Registers	523	131
Write Holding Register	506	126
Write Coil	501	125
Read DB	519	130
Write DB	530	133

Tabella 10: Composizione di training set e test set, dataset D2 e D4, classificazione multiclasse

6.2.1 Classificazione binaria

Per prima cosa verranno presentati i risultati con tutte le feature (ovvero degli esperimenti con i dataset D1 e D2), a cui seguiranno quelli con le feature rimosse (ovvero degli esperimenti con i dataset D3 e D4).

Una prima analisi dei risultati della prima coppia di esperimento è ottenibile osservando le metriche di classificazione ottenute, visionabili nelle due tabelle 11 e 12. Da queste misurazioni possiamo notare come il classificatore che sembra perfomare meglio è il KNN. Lo stacco fra KNN e gli altri è particolarmente marcato nel caso con undersampling. Il sotto-campionamento causa una generale riduzione di tutte le metriche tranne la recall, che invece tende a salire.

Classificatore	Accuratezza	Precisione	Recall	F1-Score
SVM	0.992	0.996	0.853	0.912
KNN	0.999	0.999	0.982	0.991
Naive Bayes	0.515	0.525	0.751	0.382
Albero decisionale	0.997	0.998	0.937	0.966
Random forest	0.997	0.998	0.937	0.966
MLP	0.997	0.998	0.937	0.964
Gradient Boosting	0.997	0.998	0.937	0.965
Adaboost	0.997	0.998	0.937	0.966

Tabella 11: Metriche di classificazione, classificazione binaria, dataset D1.

Classificatore	Accuratezza	Precisione	Recall	F1-Score
SVM	0.982	0.606	0.929	0.667
KNN	0.998	0.890	0.996	0.937
Naive Bayes	0.503	0.505	0.750	0.344
Albero decisionale	0.946	0.542	0.939	0.564
Random forest	0.946	0.542	0.939	0.564
MLP	0.996	0.792	0.936	0.849
Gradient Boosting	0.946	0.542	0.939	0.564
Adaboost	0.946	0.542	0.939	0.564

Tabella 12: Metriche di classificazione, classificazione binaria, dataset D2.

I risultati della cross validation sono invece quelli riportati nelle immagini 9 e 10, e da essi possiamo notare come i risultati ottenuti siano molto stabili: gli errori di cross-validation sono molto bassi. Ovviamente c'è sempre da tenere in considerazione il fatto che i risultati sono presumibilmente influenzati dallo sbilanciamento. Nel caso con sotto-campionamento l'errore è leggermente più grande per tutti i classificatori, ma rimane comunque basso. Il Naive-Bayes è quello che presenta non solo i risultati peggiori, ma anche i più instabili.

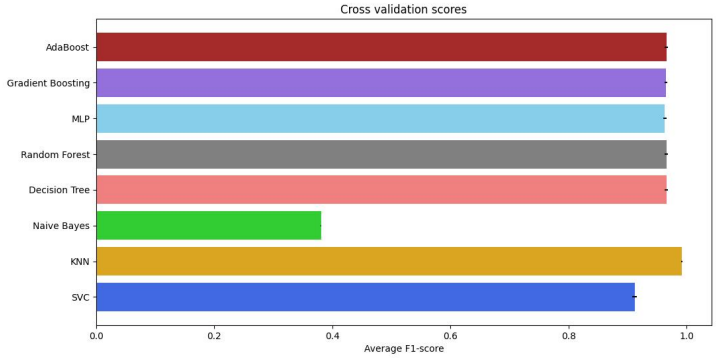

Figura 9: Risultati della cross validation, classificazione binaria, dataset D1

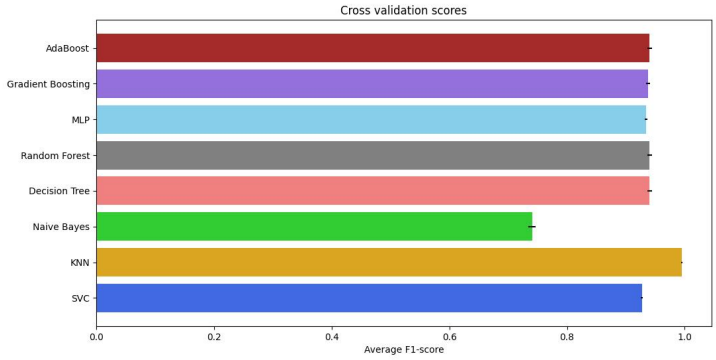

Figura 10: Risultati della cross validation, classificazione binaria, dataset D2

Per analizzare i risultati con maggiore cognizione di causa è possibile osservare le matrici di confusione. Quelle che descrivono i risultati dei test senza sotto-campionamento sono visionabili nell'immagine 11. Analizzando queste matrici possiamo notare come il KNN spicca fra tutti, come già si era visto dai valori delle metriche. È infatti l'unico a riuscire a performare molto bene su entrambe le classi, con un tasso di falsi negativi particolarmente più basso di tutti gli altri (a esclusione del Naive Bayes). Il Naive Bayes e l'SVC sono invece i peggiori, per motivi opposti. L'SVC ha infatti il più alto tasso di falsi negativi, mentre il Naive Bayes, pur essendo l'unico a non avere falsi negativi, ha un tasso di falsi positivi di circa il 50%: delle predizioni essenzialmente casuali. Tutti gli altri invece si assestano su praticamente gli stessi risultati, con performance perfette sul traffico sicuro e risultati buoni ma non buonissimi sul malevolo (tasso di falsi negativi intorno all'87%).
Le matrici di confusione del caso sotto-campionato invece sono visionabili nell'immagine 12. È chiaro come ci sia stato, come prevedibile, un generale miglioramento dei risultati per tutti. Il KNN continua a essere il migliore, questa volta con uno stacco notevole. Il tasso di veri positivi è migliorato ulteriormente, arrivando a circa il 99%. L'MLP e il Naive Bayes non vedono differenze particolarmente rilevanti. Tutti gli altri classificatori vedono invece un miglioramento per la classe del traffico malevolo, con delle generali riduzioni del tasso di falsi negativi. C'è però un costo da pagare per il miglioramento nei confronti del traffico malevolo: il tasso di falsi positivi è cresciuto generalmente per tutti. Per il KNN e l'MLP questa crescita è limitata, mentre per altri, come l'albero decisionale o il gradient boosting, questo tasso arriva anche attorno al 5%, che non è trascurabile. Si può concludere a questo punto che il miglior risultato ottenuto è stato quello del KNN con undersampling.

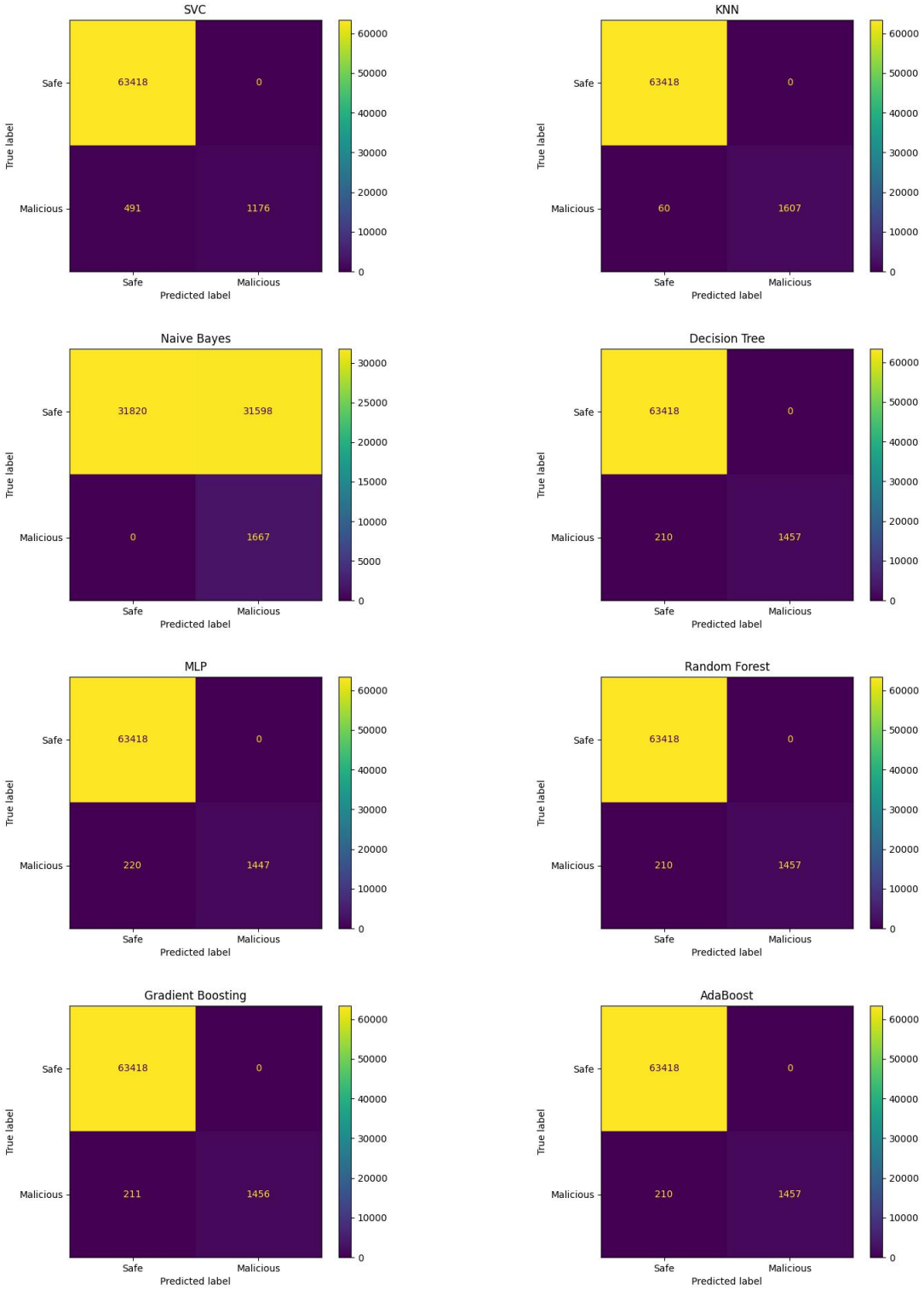

Figura 11: Matrici di confusione, classificazione binaria, dataset D1

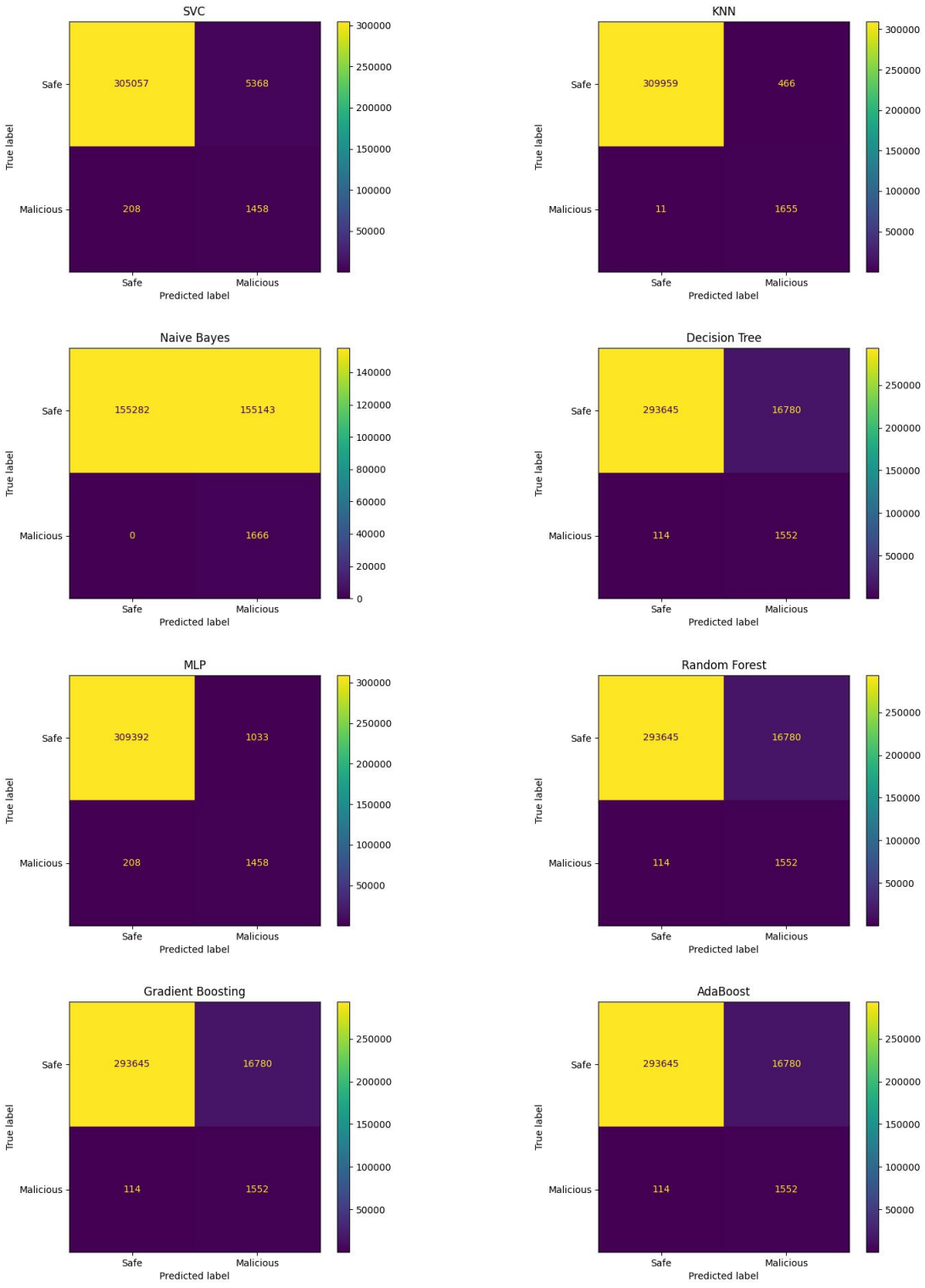

Figura 12: Matrici di confusione, classificazione binaria, dataset D2

Successivamente, vengono analizzati i risultati dei test eseguiti sui dataset D3 e D4, ovvero quelli soggetti alla rimozione delle feature. Le metriche di classificazione si trovano nelle tabelle 13 e 14. Dalla prima possiamo notare, nel caso senza sottocampionamento, un sensibile peggioramento rispetto al caso nel quale le feature di provenienza e destinazione dei pacchetti erano presenti. La situazione è più diversificata con il sotto-campionamento. Infatti, a dipendenza del modello si notano sia miglioramenti sia peggioramenti.

Classificatore	Accuratezza	Precisione	Recall	F1-Score
SVM	0.986	0.886	0.812	0.845
KNN	0.991	0.929	0.884	0.906
Naive Bayes	0.971	0.714	0.748	0.729
Albero decisionale	0.990	0.923	0.867	0.893
Random forest	0.990	0.918	0.873	0.894
MLP	0.989	0.919	0.836	0.873
Gradient Boosting	0.992	0.933	0.898	0.915
Adaboost	0.992	0.918	0.925	0.922

Tabella 13: Metriche di classificazione, classificazione binaria, dataset D3

Classificatore	Accuratezza	Precisione	Recall	F1-Score
SVM	0.942	0.537	0.908	0.554
KNN	0.991	0.683	0.933	0.756
Naive Bayes	0.972	0.548	0.746	0.575
Albero decisionale	0.995	0.756	0.933	0.822
Random forest	0.995	0.757	0.934	0.823
MLP	0.976	0.582	0.925	0.632
Gradient Boosting	0.995	0.754	0.934	0.819
Adaboost	0.990	0.668	0.932	0.741

Tabella 14: Metriche di classificazione, classificazione binaria, dataset D4

I risultati delle cross validation sono nelle figure 13 e 14. I risultati senza undersampling vedono una crescita dell'instabilità, con errori che raggiungono circa l'1% per l'SVM e l'MLP. Nell'altro caso l'instabilità è mediamente minore, eccezion fatta per il Naive-Bayes ma soprattutto per il KNN, che ora vede un'altissima variabilità dell'F1-score, con un errore di cross-validazione che raggiunge anche il 10%. Possiamo intuire da questo che il KNN potrebbe non essere la scelta più affidabile per ottenere un modello solido una volta rimosse le informazioni sugli indirizzi IP e la porta.

I risultati sulle singole classi sono quindi visionabili nelle relative matrici di confusione. Partendo dai casi senza sotto-campionamento, visionabili nell'immagine 15, si possono notare delle differenze interessanti rispetto ai casi precedenti. La capacità di riconoscere correttamente i campioni di traffico sicuro non sembra essere cambiata in maniera significativa. Al contrario, c'è una chiara

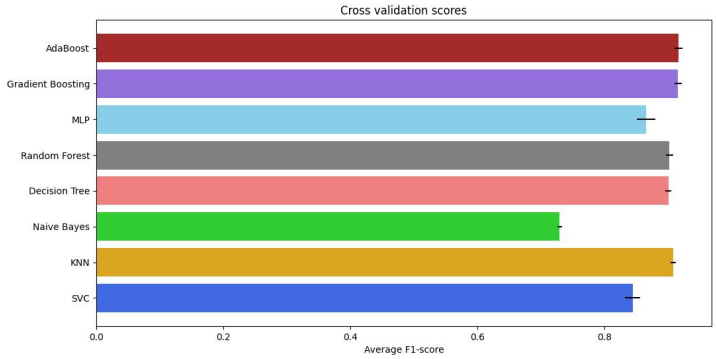

Figura 13: Risultati della cross validation, classificazione binaria, dataset D3

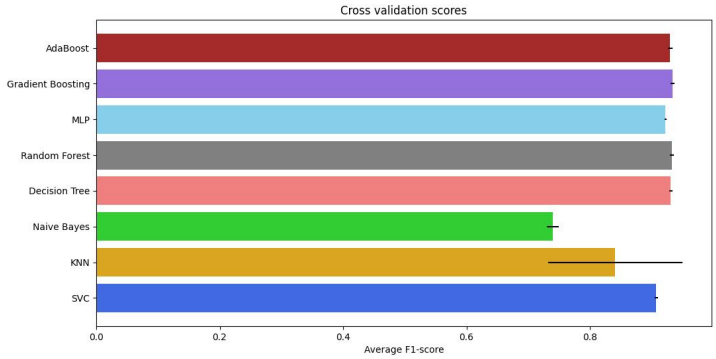

Figura 14: Risultati della cross validation, classificazione binaria, dataset D4

riduzione del tasso di veri positivi per praticamente tutti i classificatori. Nota a parte va fatta per il classificatore bayesiano, che vede i propri risultati per le due classi essenzialmente scambiate, rimanendo decisamente il peggiore di tutti.

Comportamento analogo si ritrova con il sotto-campionamento, i cui risultati sono presentati nell'immagine 16. Escluso il classificatore bayesiano infatti, tutti gli altri si assestano su un tasso di veri positivi molto simile, che si aggira attorno all'87-88%. Questo fatto sarà facilmente spiegabile una volta analizzati i risultati del caso multiclasse.

Per concludere, si può affermare che i risultati migliori si ottengono anche in questo caso con undersampling, e i migliori classificatori, considerando sia il tasso di falsi positivi sia quello di falsi negativi, sono il decision tree, il gradient boosting e la random forest.

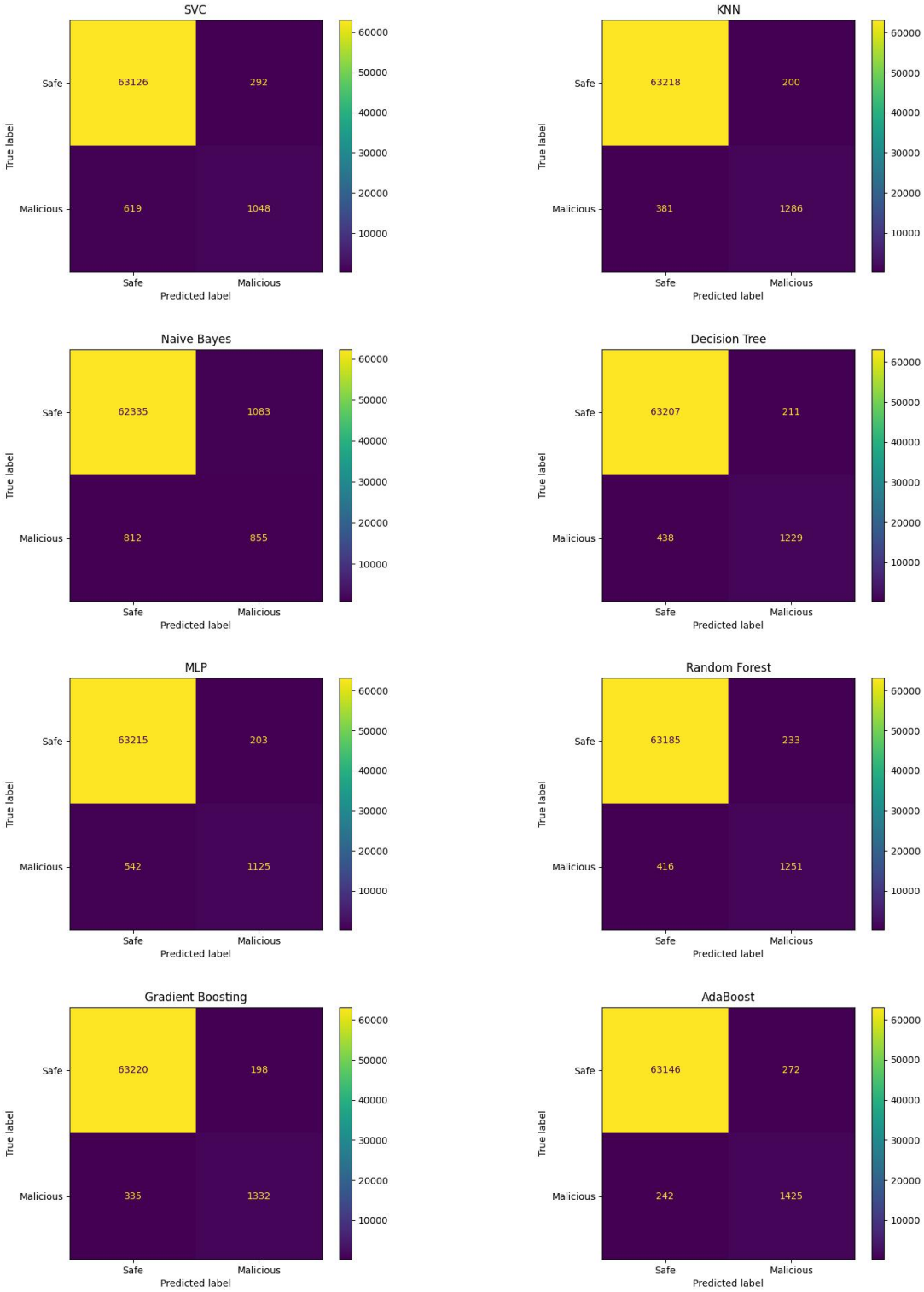

Figura 15: Matrici di confusione, classificazione binaria, dataset D3

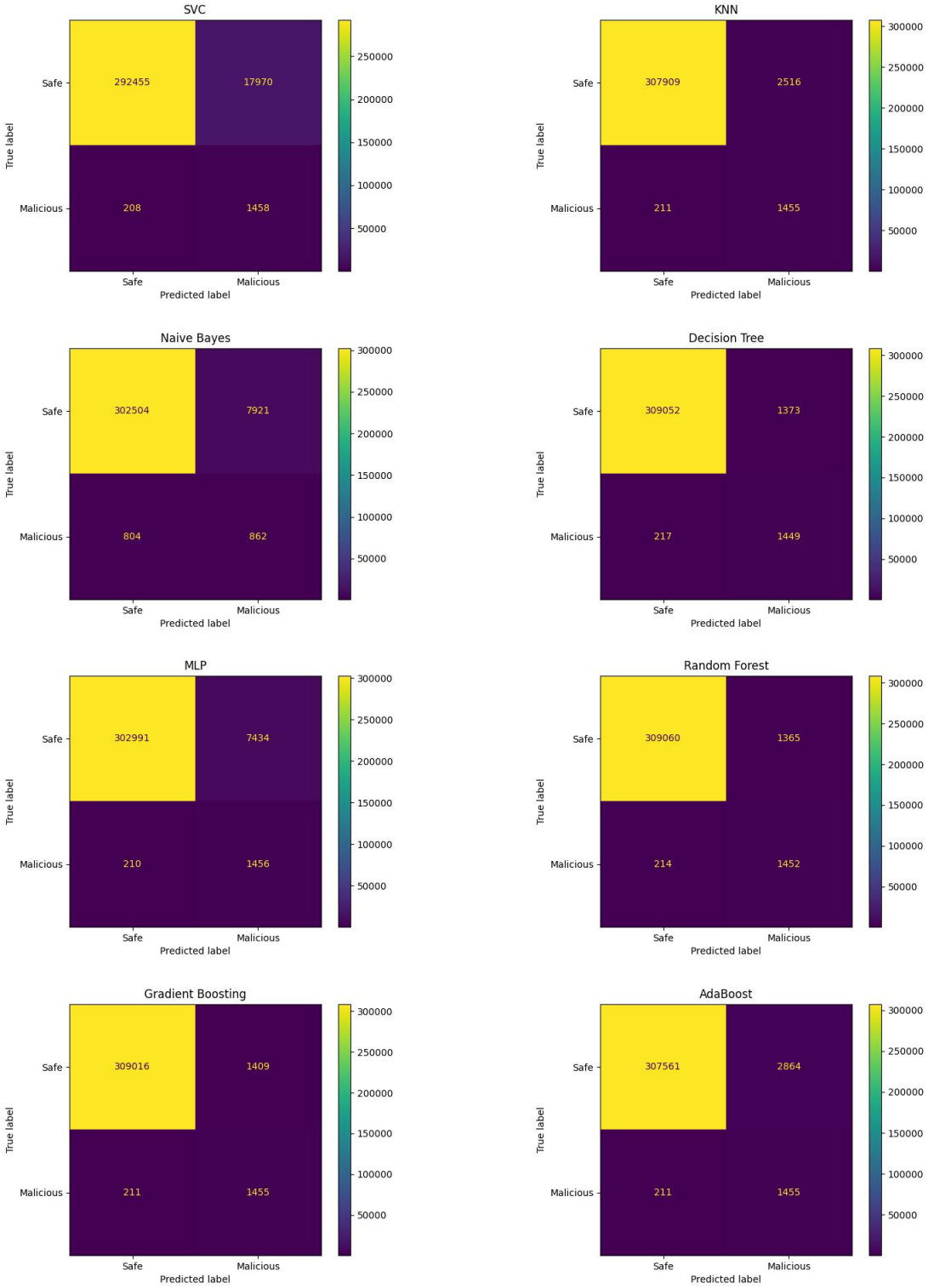

Figura 16: Matrici di confusione, classificazione binaria, dataset D4

6.2.2 Classificazione multiclasse

Vengono ora qua presentati i risultati dei test di classificazione multiclasse. Le classi sono 12, una per ciascuna classe di attacco oltre al traffico sicuro. Le metriche medie di classificazione ottenute dai classificatori lavorando con la prima coppia di dataset sono visionabili nelle tabelle 15 e 16. Al contrario del caso binario, fin da queste metriche possiamo vedere come questa volta il KNN non sia più il migliore. I classificatori con l'F1-score più alto sono infatti l'albero decisionale e la random forest. Da notare come invece l'Adaboost presenti risultati estremamente scadenti. Stavolta il sotto-campionamento migliora le metriche di alcuni classificatori e le peggiora per altri. I due classificatori più promettenti in questo caso sono la random forest e il gradient boosting.

Classificatore	Accuratezza	Precisione	Recall	F1-Score
SVM	0.988	0.583	0.568	0.566
KNN	0.997	0.916	0.891	0.901
Naive Bayes	0.734	0.852	0.974	0.854
Albero decisionale	0.997	0.915	0.916	0.916
Random forest	0.997	0.915	0.916	0.916
MLP	0.994	0.833	0.807	0.793
Gradient Boosting	0.992	0.721	0.731	0.702
Adaboost	0.976	0.127	0.168	0.140

Tabella 15: Metriche di classificazione medie, classificazione multiclasse, dataset D1

Classificatore	Accuratezza	Precisione	Recall	F1-Score
SVM	0.981	0.478	0.676	0.509
KNN	0.998	0.857	0.909	0.868
Naive Bayes	0.729	0.831	0.973	0.825
Albero decisionale	0.946	0.909	0.951	0.910
Random forest	0.946	0.915	0.951	0.914
MLP	0.997	0.831	0.899	0.845
Gradient Boosting	0.946	0.915	0.951	0.914
Adaboost	0.928	0.111	0.242	0.123

Tabella 16: Metriche di classificazione medie, classificazione multiclasse, dataset D2

I risultati della 5-fold cross validation sono visionabili nelle immagini 17 e 18. Nel caso senza undersampling alcuni classificatori presentano errori molto bassi, mentre per altri l'instabilità è non indifferente, soprattutto per il Gradient Boosting, e l'Adaboost, che raggiungono errori rispettivamente del 3% e del 4% circa. Nel caso con undersampling l'instabilità è generalmente migliore. L'errore più alto è quello dell'SVM, che è poco meno del 2%. In tutti gli altri casi sono al di sotto dell'1%, il che identifica quindi risultati piuttosto stabili.

Le matrici di confusione (immagini 19, 20, 21, 22), per motivi di spazio, utilizzano un indice nu-

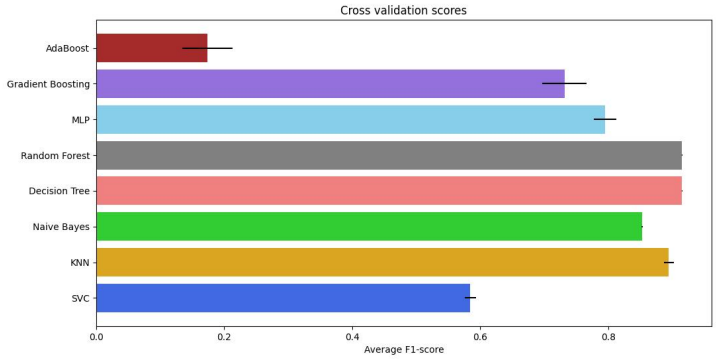

Figura 17: Risultati della cross validation, classificazione multiclasse, dataset D1

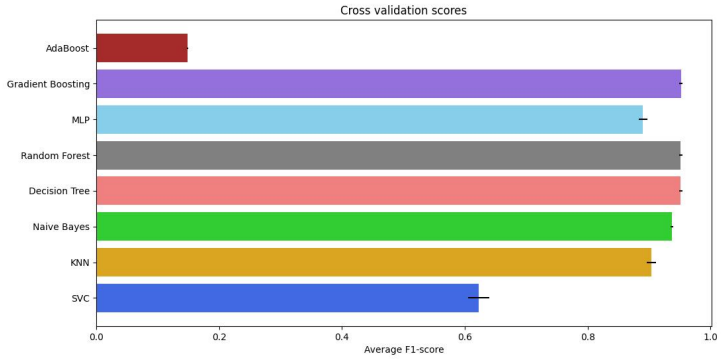

Figura 18: Risultati della cross validation, classificazione multiclasse, dataset D2

merico invece del nome delle classi. Si trovano nello stesso ordine della lista presente nelle tabelle delle proporzioni del dataset. Tramite questi risultati si possono fare importanti considerazioni. Partendo dai test senza sotto-campionamento, potrebbe sembrare che il miglior classificatore sia stato il Naive-Bayes, dato che è l'unico a riuscire a isolare quasi perfettamente tutte le classi di attacco. Tuttavia, fatica a distinguere il traffico sicuro da quello relativo allo scan dei protocolli: nello specifico, una parte sostanziosa della classe 0 (traffico standard) viene classificata come di classe 1 (scan dei protocolli). Situazione analoga ma opposta è affrontata dall'albero decisionale e dalla random forest, che fanno l'errore di classificare l'intera classe 1 come di classe 0, presentando invece risultati quasi perfetti con le altre classi. Se si analizzano questi due risultati considerando anche gli sbilanciamenti delle classi si può affermare che la random forest e l'albero decisionale sono molto meno problematici del classificatore bayesiano: categorizzare il 30% circa del traffico sicuro come malevolo è già di per sé problematico, ma considerando che a livello numerico i pacchetti sicuri sono molti di più di quelli malevoli la situazione per il bayesiano peggiora ulteriormente. A livello di effettiva utilizzabilità, è preferibile non riuscire a riconoscere i protocol scan all'avere così tanti falsi positivi per il traffico sicuro. Il KNN invece, che prima era sempre stato il migliore, qua trova difficoltà a distinguere le classi 4 e 5, ovvero la lettura delle coil e dei discrete input. In misura minore, ha dei problemi anche con le classi 1 e 6 (lettura degli holding register). Simili sono i risultati dell'MLP, che però non riesce proprio a distinguere le classi 0 e 1, e ha anche maggiori problemi con la classe 7 (lettura degli input register). Infine, Gradient boosting, Adaboost e SVM

sono quelli che hanno chiaramente avuto i risultati peggiori. Tra questi spicca l'Adaboost, che tende a classificare la maggior parte delle classi come traffico sicuro.
Analizzando invece le matrici relative agli esperimenti con undersampling, si rileva che si sono ottenuti miglioramenti per ogni classificatore, al di fuori dell'Adaboost che presenta sì cambiamenti, ma non si possono davvero definire dei miglioramenti. L'albero decisionale e la random forest riducono parzialmente i loro problemi con la classe 1, arrivando però a un tasso di predizioni corrette di solamente il 50%, ancora molto insoddisfacente. Il gradient boosting, rispetto al caso precedente, migliora moltissimo, portandosi essenzialmente alla pari dell'albero decisionale e della random forest. Il Naive Bayes rimane pressoché inalterato. Il KNN non ha quasi più problemi a distinguere le classi 0 e 1, ma rimangono problematiche le classi 4,5 e 6. Infine, l'MLP presenta miglioramenti per le classi 4 e 6, ma ancora non riesce a distinguere per nulla la 1 dalla 0.
In conclusione, nessuno dei modelli prodotti è soddisfacente al 100%, ma se si accetta di trascurare il protocol scan l'albero decisionale e la random forest offrono ottimi risultati, con e senza undersampling.

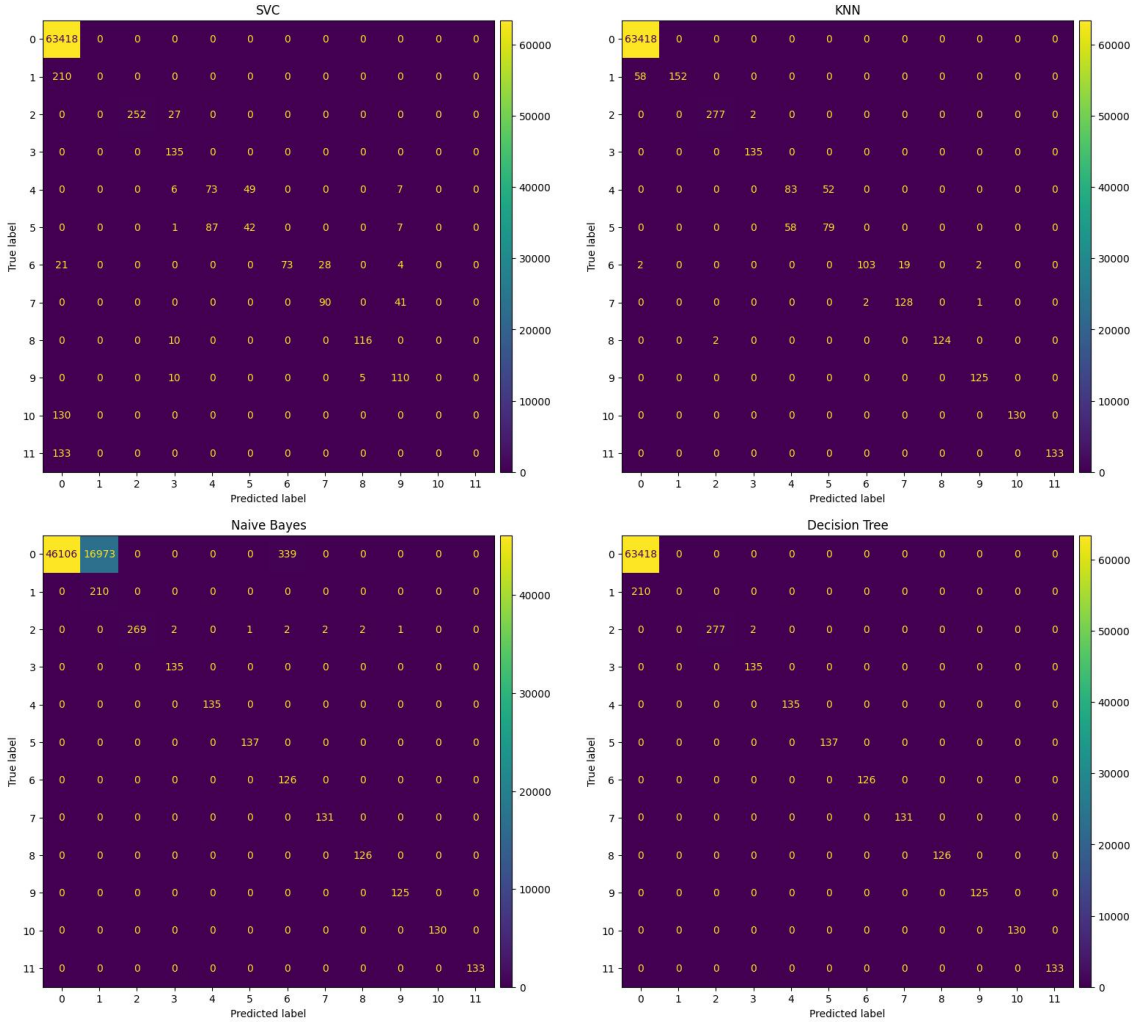

Figura 19: Matrici di confusione, classificazione multiclasse, dataset D1

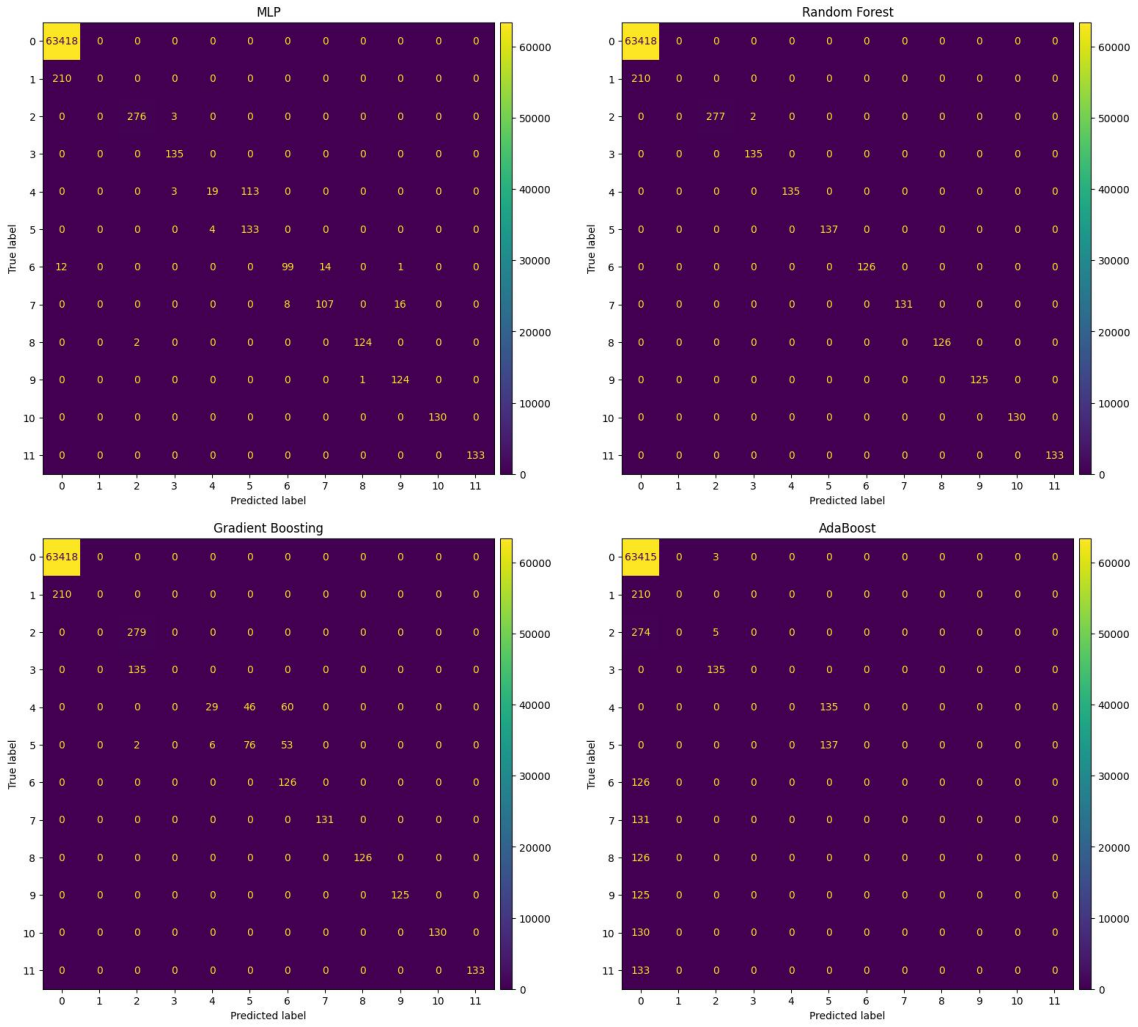

Figura 20: Matrici di confusione, classificazione multiclasse, dataset D1 (cont)

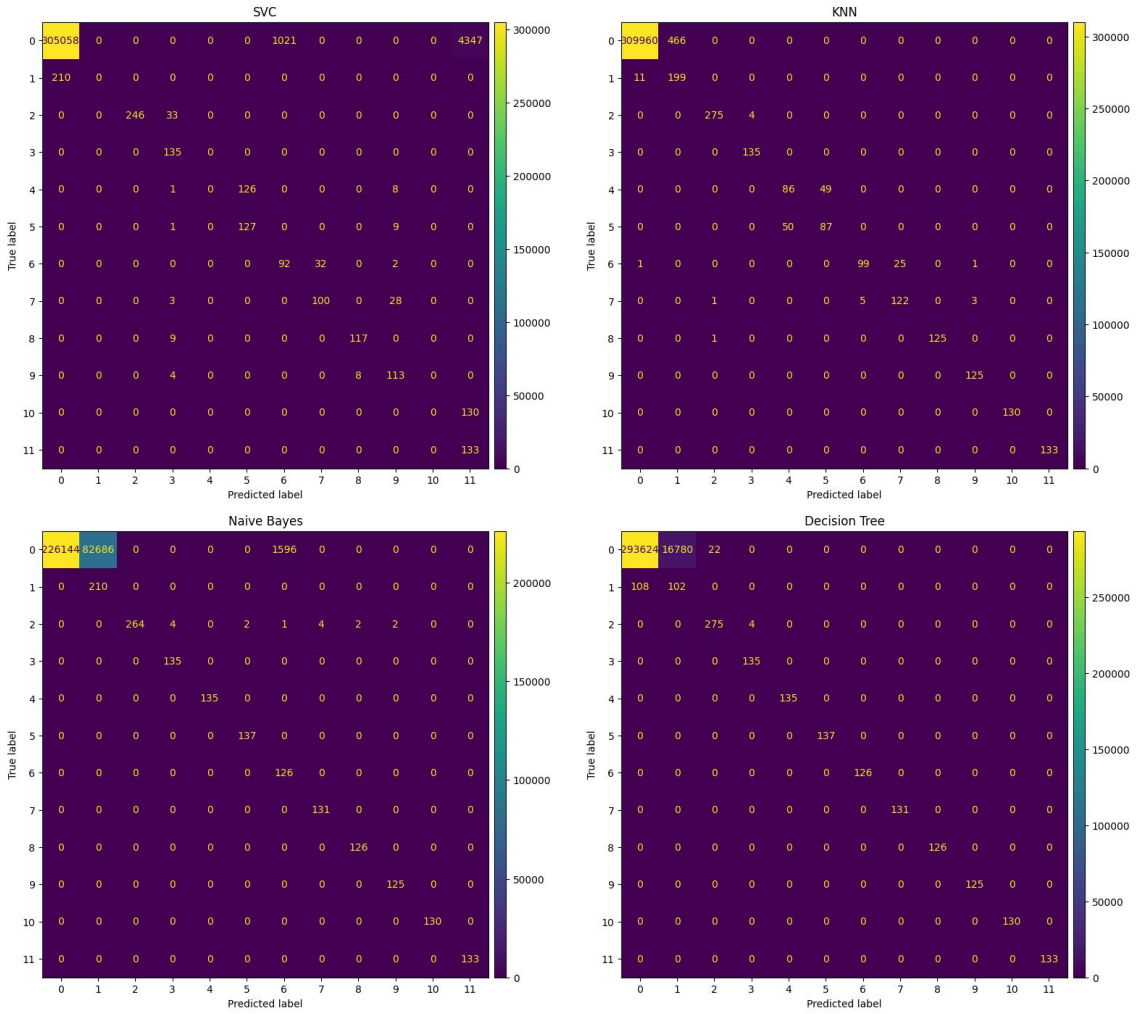

Figura 21: Matrici di confusione, classificazione multiclasse, dataset D2

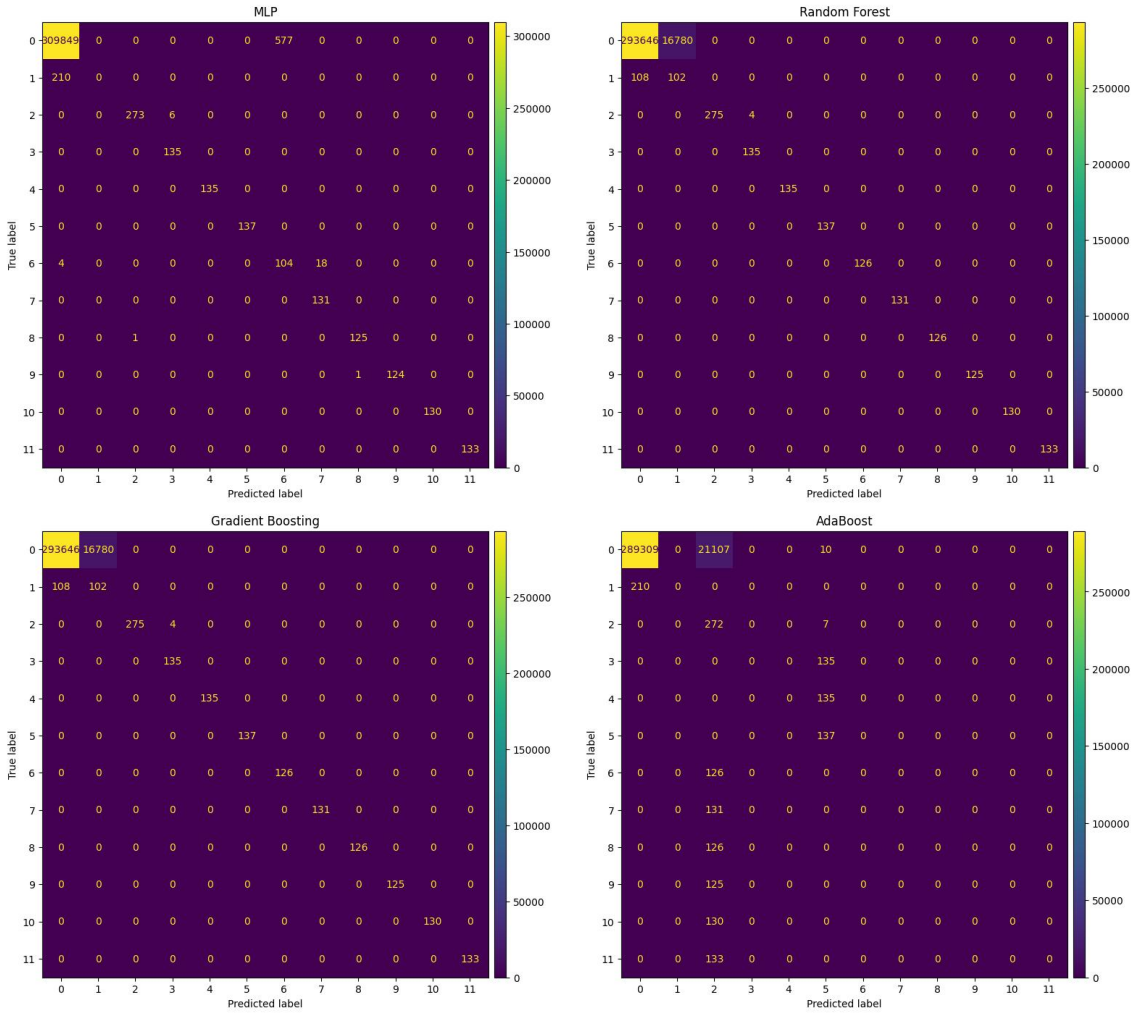

Figura 22: Matrici di confusione, classificazione multiclasse, dataset D2 (cont)

I risultati successivi sono quelli ottenuti dai classificatori lavorando senza le feature di provenienza e destinazione, ovvero coni dataset D3 e D4. Le metriche medie di classificazione, con e senza undersampling, sono visionabili nelle tabelle 17 e 18. Possiamo rilevare, rispetto ai risultati ottenuti utilizzando tutte le feature, una generale riduzione delle metriche di classificazione.

Va però prima valutata la stabilità dei modelli tramite i risultati della 5-fold cross validation.

Classificatore	Accuratezza	Precisione	Recall	F1-Score
SVM	0.983	0.583	0.547	0.559
KNN	0.989	0.719	0.721	0.719
Naive Bayes	0.446	0.577	0.949	0.638
Albero decisionale	0.989	0.767	0.766	0.767
Random forest	0.989	0.776	0.781	0.779
MLP	0.986	0.679	0.613	0.586
Gradient Boosting	0.983	0.468	0.444	0.438
Adaboost	0.977	0.127	0.168	0.140

Tabella 17: Metriche di classificazione medie, classificazione multiclasse, dataset D3

Classificatore	Accuratezza	Precisione	Recall	F1-Score
SVM	0.941	0.361	0.778	0.393
KNN	0.991	0.628	0.829	0.656
Naive Bayes	0.437	0.412	0.948	0.439
Albero decisionale	0.995	0.745	0.909	0.777
Random forest	0.995	0.747	0.913	0.779
MLP	0.989	0.624	0.892	0.669
Gradient Boosting	0.995	0.743	0.913	0.776
Adaboost	0.928	0.111	0.242	0.123

Tabella 18: Metriche di classificazione medie, classificazione multiclasse, dataset D4

L'errore di cross validazione ottenuto dai modelli allenati senza undersampling (immagine 23) è molto basso per la random forest, il decision tree, il naive-bayes e il KNN. Per gli altri l'instabilità è non indifferente. Il peggiore è il Gradient Boosting, che presenta un errore quasi del 5%. Nel caso sotto-campionato invece (immagine 24) la variabilità dei risultati è generalmente molto minore, sempre inferiore all'1%

Infine, dallo studio delle matrici di confusione è possibile, confrontando i risultati ottenuti con i

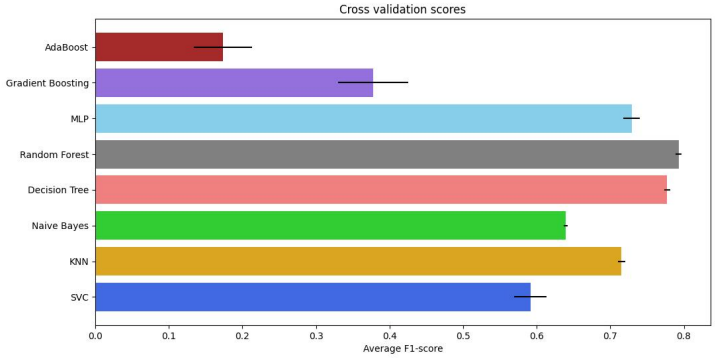

Figura 23: Risultati della cross validation, classificazione multiclasse, dataset D3

precedenti, analizzare su quali classi i classificatori hanno performato meglio o peggio rimuovendo indirizzi IP e porta di destinazione dalle feature. Quelle relative al caso senza sotto-campionamento (immagini 25, 26) mostrano dei significativi peggioramenti per tutti i classificatori (escluso l'Adaboost che rimane pessimo), cosa che suggerisce che i modelli allenati senza sotto-campionamento in precedenza si basavano sugli IP e porta per prendere alcune decisioni, in alcuni casi anche in maniera massiccia. Nello specifico, il decision tree e la foresta vedono, oltre a un leggero calo delle predizioni corrette per la classe 2, un calo molto sostanzioso per le classi 8 e 9 (le due scritture Modbus), che non riescono a distinguere dal traffico sicuro. Il KNN perde completamente la capacità di riconoscere il protocol scan (classe 1), e presenta gli stessi problemi dell'albero decisionale e della random forest per le classi 8 e 9. Mantiene poi i problemi con le classi 4,5 e 6. L'MLP, il Gradient Boosting vedono un peggioramento significativo per un grande numero di classi. Anche l'SVM vede grossi peggioramenti per la maggior parte delle classi, tranne per la 11, che riesce ora

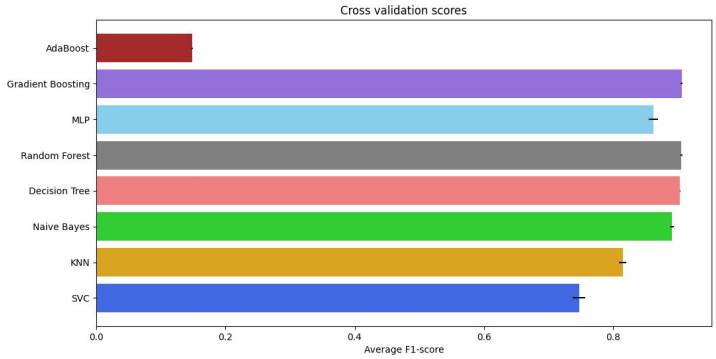

Figura 24: Risultati della cross validation, classificazione multiclasse, dataset D4

a isolare bene. Infine, il classificatore bayesiano vede una significativa riduzione della capacità di distinguere il traffico sicuro dal protocol scan: più del 50% dei campioni di traffico sicuro è infatti classificato come di classe 1. Oltre a ciò, ci sono anche dei campioni sicuri che vengono scambiati come appartenenti a varie classi di attacco.

Fortunatamente il sotto-campionamento riduce di molto le problematiche, pur non risolvendole. Dai risultati, visionabili nelle matrici presenti nelle immagini 27 e 28, se confrontati con i risultati del precedente caso con sotto-campionamento (immagini 21, 22) si rilevano, per l'appunto, peggioramenti più limitati ma comunque rilevanti (al di fuori, nuovamente, dell'Adaboost). Il decision tree, la random forest e il gradient boosting perdono le loro già molto limitate capacità di riconoscere la classe 1, che ora è completamente classificata come traffico sicuro. Per il resto, al contrario del caso senza undersampling, i classificatori si comportano molto bene, e presentano anche una riduzione significativa del tasso di falsi positivi. Anche il KNN, che con tutte le feature era l'unico classificatore che riusciva a lavorare bene sia sulla classe 0 sia sulla classe 1, in questo caso non riesce per nulla a distinguere le due, e i pacchetti di protocol scan vengono sempre classificati come di classi 0. A differenza del caso senza undersampling però, le problematiche sulle classi 8 e 9 non compaiono. L'MLP presenta dei leggerissimi peggioramenti. L'SVM mostra peggioramenti per le classi 4 e 5, mentre migliora la situazione per la 10, che ora non distingue più con la 11. Infine, il bayesiano ha un comportamento analogo a quello ottenuto senza undersampling.

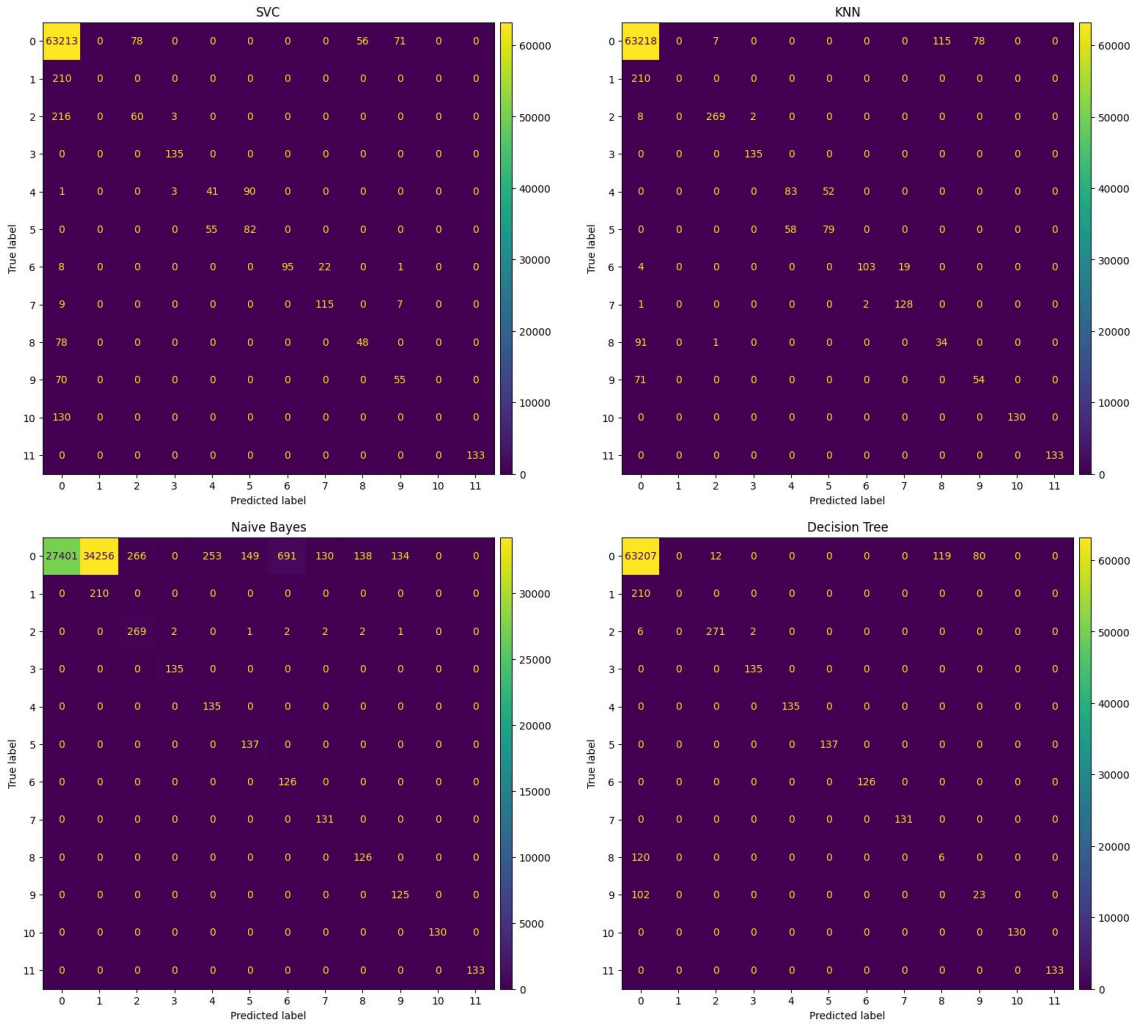

Figura 25: Matrici di confusione, classificazione multiclasse, dataset D3

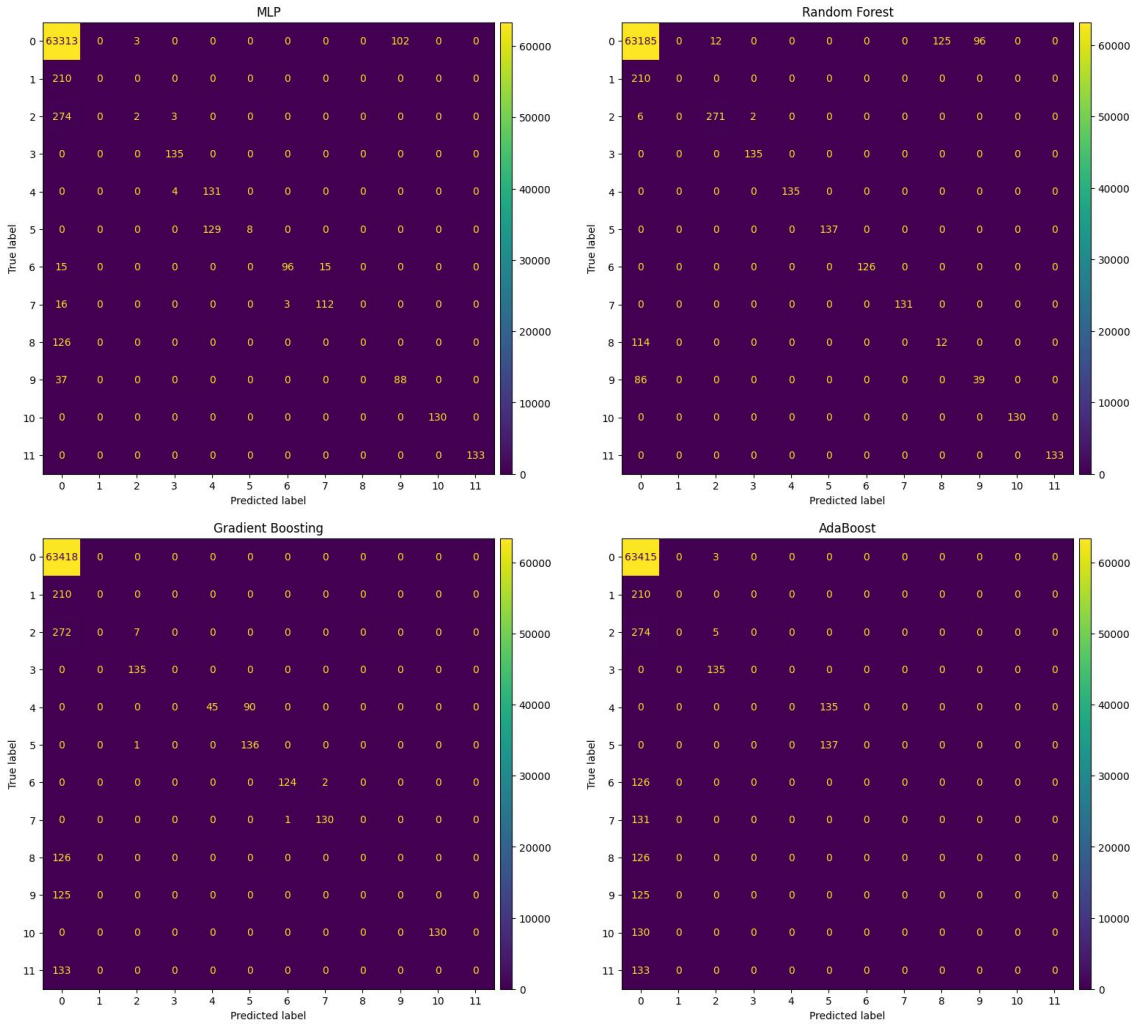

Figura 26: Matrici di confusione, classificazione multiclasse, dataset D3 (cont)

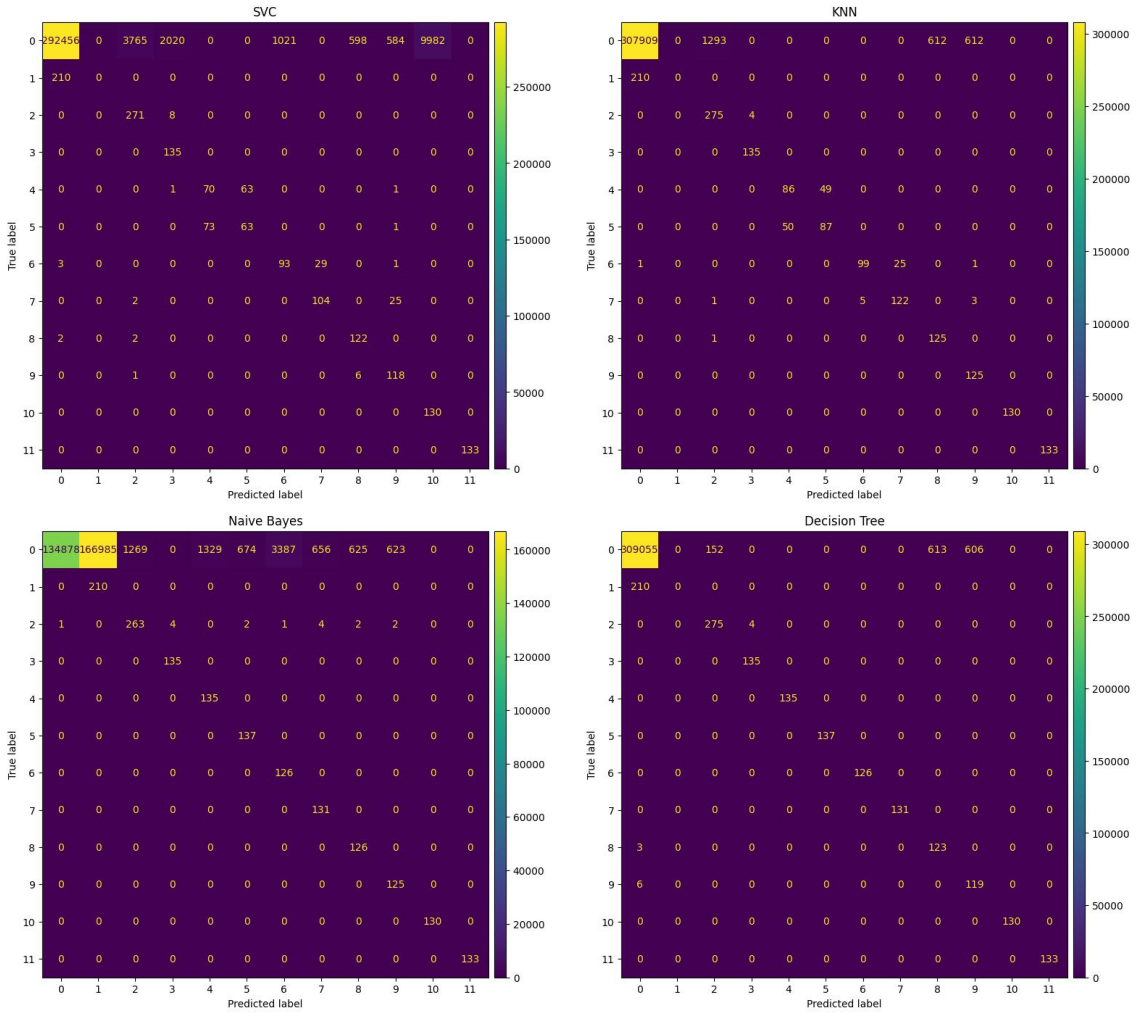

Figura 27: Matrici di confusione, classificazione multiclasse, dataset D4

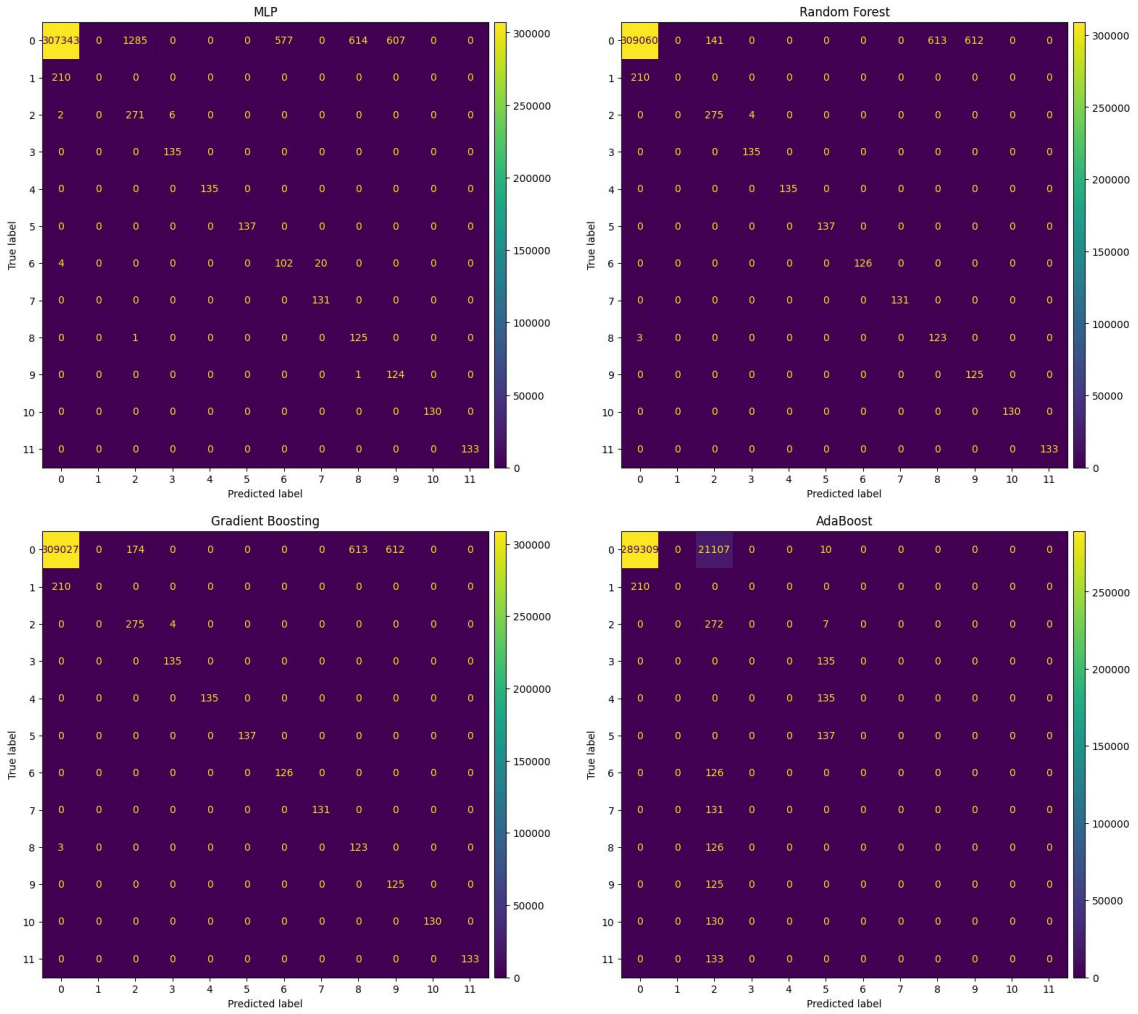

Figura 28: Matrici di confusione, classificazione multiclasse, dataset D4 (cont)

Conclusa l'analisi di tutti i risultati, sia della classificazione binaria sia di quella multiclasse si possono fare delle considerazioni. Per prima cosa, è possibile constatare che in quasi ogni singolo caso analizzato il sotto-campionamento ha migliorato i risultati: è quindi un pre-processamento che è importante fare per ridurre i forti effetti dello sbilanciamento. Per quanto riguarda invece la rimozione delle feature possiamo rilevare dei generali peggioramenti nelle performance dei classificatori. Tuttavia, questi peggioramenti non sono della stessa entità in tutti i casi: specialmente in combinazione con il sotto-campionamento alcuni classificatori, ovvero l'albero decisionale, la random forest e il gradient boosting, perdono solamente la capacità di distinguere gli elementi di classe 1 da quelli di classe 0, mantenendo intatte invece le loro capacità di lavorare con tutte le altre classi di attacco. Ragionando su che cosa rappresentano le varie classi di attacco, si può giungere alla conclusione che questa è la situazione migliore che si può raggiungere in queste condizioni di lavoro. La classe 1 rappresenta l'attacco protocol scan, e, come spiegato nella sezione 5.2.1, questo attacco consiste semplicemente nel tentativo di stabilire una connessione TCP sulla porta specifica del protocollo che si sta testando. Questo significa che, in assenza delle feature di indirizzo IP sorgente, indirizzo IP destinazione e porta di destinazione non c'è alcun modo di distinguere questi specifici pacchetti dagli altri tentativi di connessione TCP presenti nel traffico, per lo meno analizzando esclusivamente il singolo pacchetto, come è stato scelto di fare nel lavoro qui presentato.

È quindi impossibile per i classificatori imparare a distinguere le classi 0 e le classi 1, e possiamo dunque affermare che le performance dell'albero decisionale, della random forest e del Gradient Boosting, se allenate con un traning set sotto-campionato, sono assai soddisfacenti. Se invece si decide di includere tutte le feature estratte dal traffico, con la consapevolezza che si rischierebbe di arrivare all'*overfitting*, è chiaro che il miglior classificatore nel caso binario sia il KNN, mentre nel caso multiclasse ancora una volta possiamo ritenere l'albero decisionale e la random forest come le scelte migliori, pur con i limiti riscontrati sulla classe 1.

6.3 Esperimenti con i classificatori one-class

In questa sezione verranno infine presentati i risultati ottenuti dall'insieme di classificatori *one-class*. Per lavorare con questi modelli di classificazione si è sfruttata la libreria Python "PyOD" ([48]). I modelli sono stati utilizzati con i parametri di default, come nel caso dei modelli visti in precedenza, eccezion fatta per l'Average KNN, che si ottiene ponendo "mean" come valore del parametro "method" del KNN. L'Isolation Forest, l'unico modello fra quelli testati ad avere una componente di casualità, ha il seed del PNRG fissato, così come era stato fatto per i classificatori supervisionati.
Per questa famiglia è stata presa la decisione di eseguire dei test solamente con indirizzi IP e porta rimosse dalle feature, dato che si è arrivati alla conclusione che, al di fuori del protocol scan, per lo meno alcuni dei classificatori testati erano risultati in grado di gestire l'assenza di queste feature. Questo permette di stare maggiormente al sicuro dal pericolo dell'overfitting. È stato inoltre anche eseguito del sotto-campionamento. Questo, di per sé, non serviva come nei casi visti in precedenza, dato che non c'è nessuno sbilanciamento da ridurre. Ciononostante, è stato comunque eseguito, per ridurre i tempi di esecuzione. Alcuni dei classificatori testati, infatti, avrebbero richiesto un tempo molto grande se si fosse usata la maggioranza dei dati per il training. Dati i risultati ottenuti, presentati a breve, questa scelta si è dimostrata non particolarmente influente sulla qualità dei risultati. Dato che in questo caso il training set doveva essere composto esclusivamente da elementi di classe "Safe", il processo di undersampling è leggermente diverso da quello usato in precedenza. Il dataset è stato diviso in due parti, una con il traffico sicuro e una con il traffico malevolo. A questo punto, è stato eseguito uno split 80-20 del dataset "sicuro". Il 20% è stato utilizzato per il training set, mentre il restante 80% è stato unito al dataset con il traffico malevolo, formando così il test set. La composizione dei due dataset è quindi quella presentata nella tabella 19. Dopo

Dataset	Elementi di classe "Safe"	Elementi di classe "Malicious"
Training set	63418	0
Test set	253673	8332

Tabella 19: Composizione di training set e test set, esperimenti con classificatori *one-class*

l'undersampling e la rimozione delle feature, il pre-processamento si conclude con lo scaling delle feature tra 0 e 1, così come era fatto per la famiglia di classificatori precedenti.
Le metriche di classificazione calcolate una volta eseguiti i test sono presentate nella tabella 20. Da

questi risultati si può notare come un gruppo di classificatori, ovvero il One Class SVM, l'HBOS e l'Isolation Forest hanno ottenuto gli stessi identici risultati. Analogamente, anche le due versioni del One Class KNN hanno gli stessi risultati. Il solo classificatore ad avere risultati "unici" è il Feature Bagging, forse per via del fatto che il classificatore di base che sfrutta di deafult questo metodo ad *ensemble*, il Local Outlier Factor, è diverso dagli altri testati. I risultati migliori paiono essere quelli dei due KNN.

Le matrici di confusione ottenute, presentate nell'immagine 29, mostrano una situazione nella quale

Classificatore	Accuratezza	Precisione	Recall	F1-Score
One Class SVM	0.922	0.635	0.899	0.688
Feature Bagging	0.988	0.906	0.914	0.909
HBOS	0.922	0.635	0.899	0.688
Isolation Forest	0.922	0.635	0.899	0.688
One Class KNN	0.988	0.889	0.933	0.910
One Class Average KNN	0.988	0.889	0.933	0.910

Tabella 20: Metriche di classificazione, classificazione binaria "*one-class*"

tutti i classificatori, escluso il Feature Bagging, hanno ottenuto lo stesso tasso di veri positivi, che ammonta a circa l'87%. Questo valore è lo stesso che si era riusciti a raggiungere, al massimo, con i classificatori precedenti. Si può ipotizzare quindi che, analogamente al caso precedente, anche i classificatori *one-class* non riescono a riconoscere il protocol scan per mancanza di informazioni. Per analizzare più profondamente questi risultati sono state analizzate le singole predizioni ed è stata creata la tabella 21, che, per ogni classificatore e per ogni classe, identifica la percentuale di campioni correttamente classificati come sicuri o malevoli. Per motivi di spazio, i classificatori sono rappresentati da una sigla: al di fuori dell'HBOS, che usa la sua sigla classica, "OCSVM" indica il One Class SVM, "FB" il Feature Bagging, "IF" l'Isolation Forest, "OCKNN" il One Class KNN, e infine "OCAvKNN" il One Class Average KNN. I valori presenti in questa tabella confermano l'ipotesi precedente: il protocol scan è quello che dà problemi, mentre tutte le altre classi di attacco sono correttamente riconosciute come "anomale". Il Feature Bagging è l'unico a fare leggermente peggio in questo frangente, dato che si nota un po' di difficoltà per la classe 8, la scrittura degli holding register. Ciò detto, il Feature Bagging ripaga con il più basso tasso di falsi positivi. Questo tasso è leggermente più grande per i due KNN, mentre è sensibilmente maggiore per gli altri tre classificatori. Considerando sia i campioni positivi sia quelli negativi, si può affermare che il KNN sia il modello *one-class* più adatto a lavorare con il dataset testato, così come avevano preannunciato le metriche viste in precedenza. Non sono state rilevate differenze tra le due versione del One Class KNN.

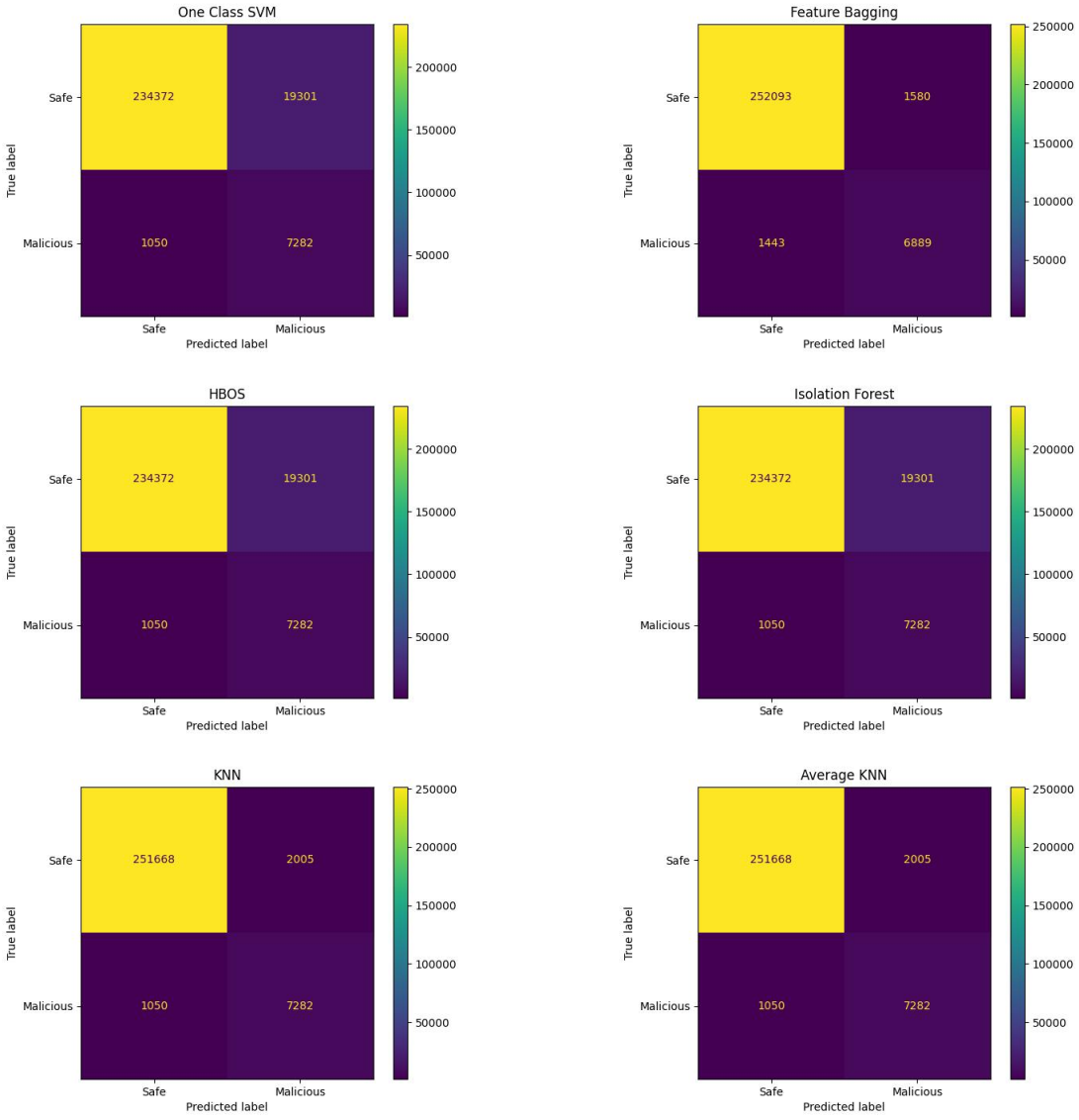

Figura 29: Matrici di confusione, classificazione binaria "*one-class*"

Classe	OCSVM	FB	HBOS	IF	OCKNN	OCAvKNN
Safe	92%	99,4%	92%	92%	99,2%	99,2%
Protocol Scan	0%	0%	0%	0%	0%	0%
Function Scan	100%	100%	100%	100%	100%	100%
UID Scan	100%	100%	100%	100%	100%	100%
Read Coils	100%	100%	100%	100%	100%	100%
Read Discrete Inputs	100%	100%	100%	100%	100%	100%
Read Holding Registers	100%	99,8%	100%	100%	100%	100%
Read Input Registers	100%	100%	100%	100%	100%	100%
Write Holding Registers	100%	79%	100%	100%	100%	100%
Write Coil	100%	100%	100%	100%	100%	100%
Read DB	100%	100%	100%	100%	100%	100%
Write DB	100%	100%	100%	100%	100%	100%

Tabella 21: Tassi di predizione corretta per ogni classe, classificatori *one-class*

7 Conclusione e sviluppi futuri

Dalle analisi presentate in questo elaborato è risultato chiaro quanto importante sia considerare la sicurezza informatica nell'ambito industriale. Esso è un ambiente sempre più allettante per i criminali informatici, per via della quantità di danni che è possibile causare e grazie al fatto che gli attacchi sono diventati più semplici da eseguire rispetto a quando le reti industriali erano essenzialmente sconnesse dal resto del mondo. Esistono diverse strategie che si possono seguire per cercare di difendersi il più possibile, e vari strumenti che si possono implementare. Uno di questi sono sicuramente gli IDS, qui discussi ampiamente. Ci sono vari approcci che possono essere seguiti per implementare un *intrusion detection system*, ciascuno con i suoi pro e i suoi contro. Gli esperimenti presentati nell'elaborato sono stati condotti in una situazione specifica: la produzione di un dataset a partire da una rete reale e in produzione, e il test di vari classificatori con dati rappresentativi di singoli pacchetti. Dati i risultati di questi test, esaminati nelle sezioni 6.2.1, 6.2.2 e 6.3, si può concludere che utilizzando tutte le feature a disposizione è possibile costruire un modello di classificazione binaria che presenta risultati molto soddisfacenti sfruttando il KNN. Questo però apre la porta alla possibilità di overfitting, che andrebbe testata con ulteriori esperimenti. D'altra parte, si è arrivati alla conclusione che, anche rimuovendo le feature che con maggiore probabilità possono causare overfitting, i classificatori binari sono comunque risultati capaci di lavorare egregiamente con tutte le classi di attacco, a esclusione del protocol scan. Questa considerazione vale anche per i classificatori *one-class*. Al contrario, non si è mai riusciti a produrre un modello di classificazione multiclasse che si comportasse molto bene su tutte le classi di attacco, neanche con tutte le feature a disposizione. Tuttavia, analizzando più a fondo i risultati si è notato che per la maggior parte dei classificatori la classe di attacco che risultava maggiormente problematica era, così come nel caso binario, quella del protocol scan. In assenza di tale classe di attacco, anche alcuni dei classificatori multiclasse testati, come per esempio la random forest, sarebbero capaci di fornire dei risultati molto buoni, paragonabili a quelli ottenuti dai classificatori binari. È quindi possibile concludere che, fintanto che le feature estratte dal traffico sono relative a un singolo panchetto, così come si è scelto di fare per questi test, il protocol scan è un attacco che sarebbe meglio non tenere in considerazione fin dal principio.
Infine, è possibile evidenziare varie strade percorribili per proseguire e migliorare le analisi qui eseguite:

- Sugli algoritmi testati non è stato eseguito fine tuning, ma questo potrebbe portare a dei miglioramenti più o meno importanti dei risultati.
- Ci sono ulteriori algoritmi che potrebbero essere testati. Per esempio, potrebbe essere interessante studiare il comportamento di algoritmi di clustering, confrontando così le loro capacità con gli algoritmi di classificazione.
- Dei test importanti che andrebbero fatti sarebbero quelli sull'overfitting dei classificatori nei casi in cui si utilizzano le feature di provenienza e destinazione dei pacchetti.

- Un altro studio estremamente rilevante è quello dello sviluppo e test sul campo di un vero e proprio IDS. Il lavoro qui presentato si è fermato al test dei classificatori che potrebbero essere utilizzabili per la sua costruzione. Tuttavia, come è stato descritto nella sezione 2.2, ci sono molte altre caratteristiche che un buon IDS dovrebbe avere oltre che le capacità di classificare correttamente il traffico.

- Infine, si potrebbe uscire parzialmente dal contesto in cui si è lavorato in questo caso, e produrre feature di tipo differente a partire dal dataset prodotto. Per esempio, sarebbe possibile costruire delle feature che identificano un flusso di pacchetti e non un singolo pacchetto. Questo permetterebbe di avere una visione più ampia del traffico, e di evidenziare comportamenti malevoli che a livello di singolo pacchetto sono di difficile rilvazione. Ne è un esempio proprio la difficoltà riscontrata con la classe di attacco protocol scan.

Riferimenti bibliografici

[1] M. Alanazi, A. Mahmood, and M. J. M. Chowdhury. Scada vulnerabilities and attacks: A review of the state-of-the-art and open issues. *Computers & Security*, page 103028, 2022.

[2] M. Alkasassbeh and S. Al-Haj Baddar. Intrusion detection systems: A state-of-the-art taxonomy and survey. *Arabian Journal for Science and Engineering*, pages 1–44, 2022.

[3] S. D. Anton, M. Gundall, D. Fraunholz, and H. D. Schotten. Implementing scada scenarios and introducing attacks to obtain training data for intrusion detection methods. In *ICCWS 2019 14th International Conference on Cyber Warfare and Security: ICCWS 2019*, page 56. Academic Conferences and publishing limited, 2019.

[4] S. D. D. Anton, S. Sinha, and H. D. Schotten. Anomaly-based intrusion detection in industrial data with svm and random forests. In *2019 International conference on software, telecommunications and computer networks (SoftCOM)*, pages 1–6. IEEE, 2019.

[5] A. Bécue, I. Praça, and J. Gama. Artificial intelligence, cyber-threats and industry 4.0: Challenges and opportunities. *Artificial Intelligence Review*, 54(5):3849–3886, 2021.

[6] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[7] Dake and Mysid. CC BY 1.0 `https://creativecommons.org/licenses/by/1.0`, via Wikimedia Commons, convertito in formato PNG. Accessed: 2023-12-14.

[8] A. Dehlaghi-Ghadim, M. H. Moghadam, A. Balador, and H. Hansson. Anomaly detection dataset for industrial control systems. *arXiv preprint arXiv:2305.09678*, 2023.

[9] Electra dataset: Anomaly detection ics dataset. `http://perception.inf.um.es/ICS-datasets/`. Accessed: 2023-06-06.

[10] Simatic et 200sp, siemens. `https://www.siemens.com/global/en/products/automation/systems/industrial/io-systems/et-200sp.html`. Accessed: 2023-22-12.

[11] farzinenddo. smod. `https://github.com/theralfbrown/smod-1`. Accessed: 2024-01-10.

[12] R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

[13] J. Gao, L. Gan, F. Buschendorf, L. Zhang, H. Liu, P. Li, X. Dong, and T. Lu. Omni scada intrusion detection using deep learning algorithms. *IEEE Internet of Things Journal*, 8(2):951–961, 2020.

[14] W. Gao, T. Morris, B. Reaves, and D. Richey. On scada control system command and response injection and intrusion detection. In *2010 eCrime Researchers Summit*, pages 1–9. IEEE, 2010.

[15] Á. L. P. Gómez, L. F. Maimó, A. H. Celdrán, F. J. G. Clemente, C. C. Sarmiento, C. J. D. C. Masa, and R. M. Nistal. On the generation of anomaly detection datasets in industrial control systems. *IEEE Access*, 7:177460–177473, 2019.

[16] C. S. Inc. Modbus 101 - introduction to modbus. `https://www.csimn.com/CSI_pages/Modbus101.html`. Accessed: 2023-09-09.

[17] Controller jace 8000, tridium. `https://www.tridium.com/it/it/Products/niagara/jace-8000`. Accessed: 2023-22-12.

[18] A. Keliris, H. Salehghaffari, B. Cairl, P. Krishnamurthy, M. Maniatakos, and F. Khorrami. Machine learning-based defense against process-aware attacks on industrial control systems. In *2016 IEEE International Test Conference (ITC)*, pages 1–10. IEEE, 2016.

[19] I. A. Khan, D. Pi, Z. U. Khan, Y. Hussain, and A. Nawaz. Hml-ids: A hybrid-multilevel anomaly prediction approach for intrusion detection in scada systems. *IEEE Access*, 7:89507–89521, 2019.

[20] KimiNewt. pyshark, python wrapper for tshark. `https://github.com/KimiNewt/pyshark/`. Accessed: 2024-01-10.

[21] A. Lemay, J. Fernandez, and S. Knight. An isolated virtual cluster for scada network security research. In *1st International Symposium for ICS & SCADA Cyber Security Research 2013 (ICS-CSR 2013) 1*, pages 88–96, 2013.

[22] A. Lemay and J. M. Fernandez. Providing scada network data sets for intrusion detection research. In *CSET@ USENIX Security Symposium*, 2016.

[23] O. Linda, T. Vollmer, and M. Manic. Neural network based intrusion detection system for critical infrastructures. In *2009 international joint conference on neural networks*, pages 1827–1834. IEEE, 2009.

[24] miguelob. Ics and plc pentesting and hacking. `https://github.com/miguelob/ICS-Hacking`. Accessed: 2024-01-10.

[25] G. Miru. The siemens s7 communication - part 1 general structure. `http://gmiru.com/article/s7comm/`. Accessed: 2023-09-09.

[26] G. Miru. The siemens s7 communication - part 2 job requests and ack data. `http://gmiru.com/article/s7comm-part2/`. Accessed: 2023-09-09.

[27] T. Morris and W. Gao. Industrial control system traffic data sets for intrusion detection research. In *Critical Infrastructure Protection VIII: 8th IFIP WG 11.10 International Conference, ICCIP 2014, Arlington, VA, USA, March 17-19, 2014, Revised Selected Papers 8*, pages 65–78. Springer, 2014.

[28] T. H. Morris, Z. Thornton, and I. Turnipseed. Industrial control system simulation and data logging for intrusion detection system research. *7th annual southeastern cyber security summit*, pages 3–4, 2015.

[29] D. Nardella. Snap7. `https://snap7.sourceforge.net/`. Accessed: 2023-09-10.

[30] The opc unified architecture. `https://opcfoundation.org/about/opc-technologies/opc-ua/`. Accessed: 2023-10-11.

[31] M. Organization. Modbus application protocol specification v1.1b3. `https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf`. Accessed: 2023-09-09.

[32] M. Organization. Modbus messaging on tcp/ip implementation guide v1.0b. `https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf`. Accessed: 2023-09-09.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] B. Phillips, E. Gamess, and S. Krishnaprasad. An evaluation of machine learning-based anomaly detection in a scada system using the modbus protocol. In *Proceedings of the 2020 ACM Southeast Conference*, pages 188–196, 2020.

[35] N. R. Rodofile, T. Schmidt, S. T. Sherry, C. Djamaludin, K. Radke, and E. Foo. Process control cyber-attacks and labelled datasets on s7comm critical infrastructure. In *Information Security and Privacy: 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part II 22*, pages 452–459. Springer, 2017.

[36] L. Rosa, T. Cruz, M. B. de Freitas, P. Quitério, J. Henriques, F. Caldeira, E. Monteiro, and P. Simões. Intrusion and anomaly detection for the next-generation of industrial automation and control systems. *Future Generation Computer Systems*, 119:50–67, 2021.

[37] L. Rosa, T. Cruz, M. B. de Freitas, P. Quitério, J. Henriques, F. Caldeira, E. Monteiro, and P. Simões. Intrusion and anomaly detection for the next-generation of industrial automation and control systems. *Future Generation Computer Systems*, 119:50–67, 2021.

[38] Simatic s7-1200, siemens. `https://www.siemens.com/it/it/prodotti/automazione/systems/industrial/simatic-controller/simatic-s7-1200.html`. Accessed: 2023-22-12.

[39] Simatic s7-1500, siemens. `https://www.siemens.com/it/it/prodotti/automazione/systems/industrial/simatic-controller/simatic-s7-1500.html`. Accessed: 2023-22-12.

[40] Wireshark - s7comm. `https://wiki.wireshark.org/S7comm`. Accessed: 2023-09-10.

[41] Scapy. `https://scapy.net/`. Accessed: 2024-01-10.

[42] Industrial control system (ics) cyber attack datasets. `https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets`. Accessed: 2023-06-06.

[43] snort.org. `https://www.snort.org/`. Accessed: 2023-04-10.

[44] J. Song, H. Takakura, Y. Okabe, D. Inoue, M. Eto, and K. Nakao. A comparative study of unsupervised anomaly detection techniques using honeypot data. *IEICE transactions on information and systems*, 93(9):2544–2554, 2010.

[45] suricata.io. `https://suricata.io/`. Accessed: 2023-04-10.

[46] E. Vasilomanolakis, S. Srinivasa, C. G. Cordero, and M. Mühlhäuser. Multi-stage attack detection and signature generation with ics honeypots. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1227–1232. IEEE, 2016.

[47] zeek.org. `https://zeek.org/`. Accessed: 2023-04-10.

[48] Y. Zhao, Z. Nasrullah, and Z. Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.