Alma Mater Studiorum · Università di Bologna

DIPARTIMENTO DI INFORMATICA

Progetto d'esame

Classificazione multiclasse di cyber-attack

Data Analysis

Presentata da:
Benedetta Bottari
Milena Mazza

Anno Accademico 2024/2025

Indice

Introduzione

Negli ultimi anni, la crescente complessità e diffusione degli attacchi informatici ha reso indispensabile l'adozione di strumenti avanzati per il loro rilevamento e classificazione. L'analisi del traffico di rete rappresenta una delle strategie più efficaci per individuare potenziali minacce e garantire la sicurezza dei sistemi informatici.
Questo progetto si concentra sulla classificazione multiclasse degli attacchi informatici seguendo tutte le fasi della data analysis, con l'obiettivo di distinguere il traffico malevolo da quello legittimo attraverso l'analisi dei pacchetti di rete. Per questo scopo, è stato utilizzato un dataset che include 9 diverse tipologie di attacchi informatici e una classe denominata "Normal", che rappresenta il traffico non malevolo.
Per affrontare il problema della classificazione, sono stati sperimentati diversi algoritmi di Machine Learning e Deep Learning, tra cui Support Vector Machines (SVM), Random Forest e k-Nearest Neighbors (KNN), oltre a modelli neurali avanzati come Feedforward Neural Networks, TabTransformer e TabNet.
L'analisi dei risultati ha evidenziato la superiorità delle reti neurali rispetto ai metodi tradizionali, con un'accuratezza massima del 97.91%. Questo dato conferma la capacità delle architetture neurali di apprendere rappresentazioni complesse dei dati e migliorare significativamente l'efficacia della classificazione degli attacchi informatici.

Capitolo 1

Metodologia

1.1 Data Acquisition e Data Visualization

Il dataset in esame raccoglie dati sulle connessioni di rete, creato per supportare la valutazione dell'accuratezza e dell'efficienza di diverse soluzioni di cybersecurity.
La raccolta dei dati è avvenuta all'interno di un ambiente altamente realistico, un laboratorio di grande dimensione progettato presso il *Cyber Range and IoT Labs della School of Engineering and Information Technology (SEIT) dell'Università UNSW Canberra* e dell'*Australian Defence Force Academy (ADFA)*. Questo ambiente ha permesso di generare un flusso di traffico di rete che rispecchia le situazioni reali, utilizzando fonti di dati eterogenee, con un forte focus sulla raccolta di informazioni dal traffico di rete stesso.

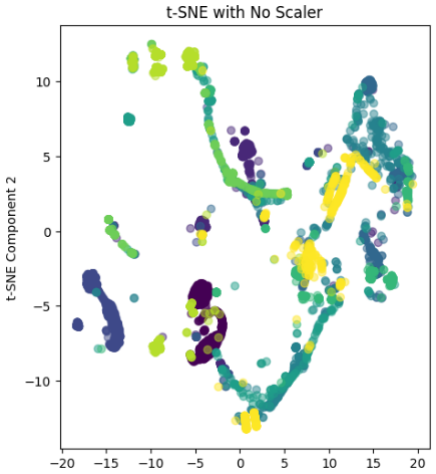


Fig. 1.1: Visualizzazione di una porzione del dataset mediante la tecnica di t-sne

Il dataset è costituito da 47 colonne, che raccolgono informazioni suddivisibili in vari gruppi, in base al tipo di dato che contengono, ad esempio sono presenti 8 colonne relative al protocollo DNN. Le variabili del dataset sono di diverso tipo: ci sono variabili numeriche, che possono essere discrete o continue, e variabili categoriche, che includono sia categorie binarie che multiclasse.
Per quanto riguarda l'aspetto della classificazione, il dataset offre due colonne principali da utilizzare come target: la colonna "Label", che si presta per una classificazione binaria, separando le attività malevole e non malevole; e la colonna "Type", che consente di realizzare una classificazione multiclasse, in grado di identificare fino a nove tipi diversi di attacchi, oltre alla classe delle attività normali, non malevole.

Prima di procedere con l'analisi dei dati, è stata effettuata una verifica preliminare, che ha rivelato che il dataset non contiene valori nulli né duplicati. Tuttavia, sono stati individuati diversi valori rappresentati dal simbolo "-", il che suggerisce la possibilità che questi rappresentino valori di default o mancanze nei dati, indicando una certa sparsità nei dati.

Colonne di attività delle connessioni

Ts

Questa colonna non presenta una correlazione logica con il tipo di attacco e, pertanto, può rappresentare un fattore di rischio per l'overfitting del modello. Le misurazioni coprono l'intervallo temporale compreso tra il 2 aprile 2019 alle ore 09:46:02 e il 29 aprile 2019 alle ore 14:45:56.

ts

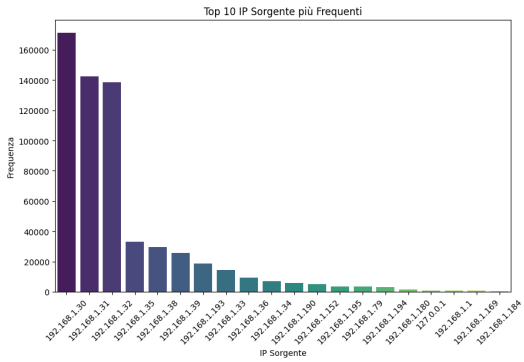
Data e ora in cui è stato catturato il pacchetto (timestamp)
Tipo colonna: Numerico Discreto - **Min**: 1554198362, **Max**: 1556549156, **Range**: 2350794
Valori di default: 0 (0.00%)
Media: 1556170825.25 , **Moda**: 1556088713, **Mediana**: 1556209864.0

Src_ip e **dst_ip**:

Entrambe le colonne presentano problemi di inconsistenza, infatti src_ip contiene 82 indirizzi in formato IPv6 e 736 indirizzi in formato IPv4, mentre dst_ip 96 indirizzi IPv6 e 735 IPv4. Inoltre src_ip , è presente il valore 0.0.0.0, che potrebbe essere associato a indirizzi non assegnati, configurazioni iniziali o condizioni di errore. Infine la distribuzione della frequenza dei diversi indirizzi non è uniforme, con alcuni indirizzi significativamente più ricorrenti di altri.

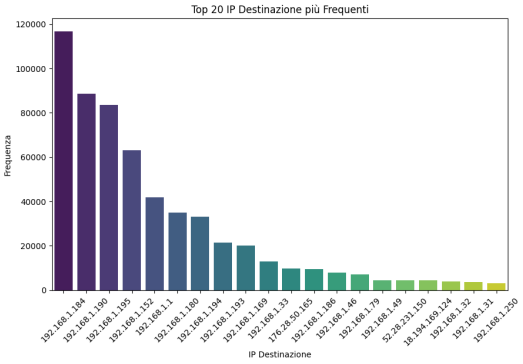
src_ip

Indirizzo IP sorgente
Tipo colonna: Categorico, **Valori unici**: 818 - ['192.168.1.152', '192.168.1.190', ...]
Valori di default: 0 (0.00%)
Moda (escluso valori di default): 192.168.1.30



dst_ip

Indirizzo IP destinazione
Tipo colonna: Categorico, **Valori unici**: 1678 ['192.168.1.152', '199.253.183.183', ...]
Valori di default: 0 (0.00%)
Moda (escluso valori di default): 192.168.1.184



Src_port e **dst_port**:

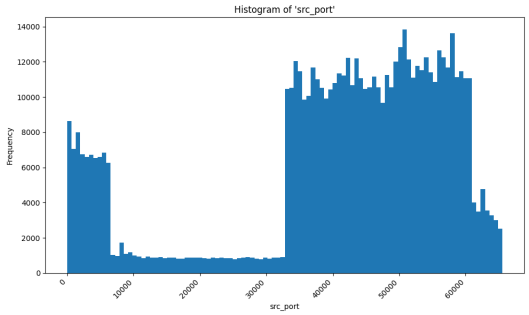
Entrambe le colonne presentano un totale elevato di valori unici, ognuno dei quali rientra nel range accettabile (0-65535) definito per i protocolli TCP/UDP, risultando quindi coerenti con il contesto. È tuttavia presente il valore 0 (8 occorrenze in src_port e 321 in dst_port), il che rappresenta un'anomalia, poiché la porta 0 non dovrebbe essere normalmente utilizzata per connessioni di rete. Questo valore potrebbe essere dovuto a errori, configurazioni particolari o impostazioni predefinite inserite durante la creazione del database. Tuttavia, può anche apparire in contesti di attacchi informatici, rendendone difficile l'interpretazione come valore di default.

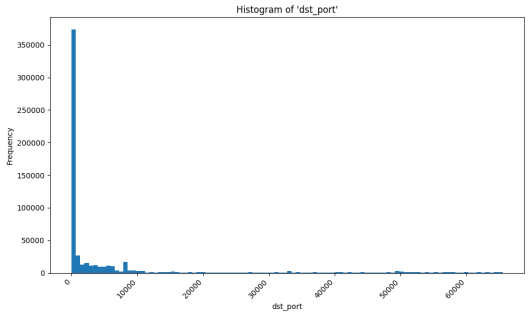
src_port

Porta sorgente utilizzata per la connessione.
Tipo colonna: Categorico
Valori unici: 57822 - [53972, 37513, 2077, ...]
Valori di default: 0 (0.00%)
Moda (escluso valori di default): 1880

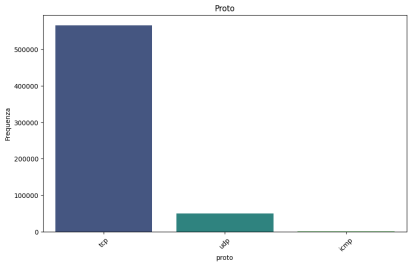
dst_port

Porta destinazione utilizzata per la connessione.
Tipo colonna: Categorico
Valori unici: 47556 - [10502, 53, 2077, ...]
Valori di default: 0 (0.00%)
Moda (escluso valori di default): 80





Proto:
La colonna presenta tre valori distinti con una distribuzione fortemente sbilanciata.



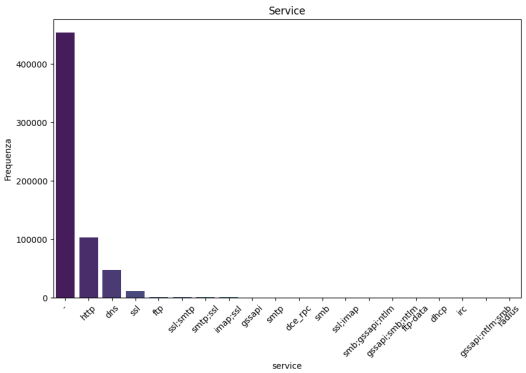
proto

Protocollo di rete utilizzato per la connessione
Tipo colonna: Categorico
Valori unici: 3 - ['tcp', 'udp', 'icmp']
Valori di default: 0 (0.00%)
Moda (escluso valori di default): tcp

Service:
Non presenta valori nettamente distinti, poiché alcuni elementi sono combinazioni di più servizi, segnalando l'uso simultaneo di più protocolli all'interno di una stessa connessione (ad esempio, `gssapi;ntlm;smb` o `ssl;smtp` per una connessione sicura di posta elettronica). Sebbene siano rilevati 23 valori unici, si può affermare che i singoli servizi distinti siano solamente 14 (`dns`, `http`, `ssl`, `ftp-data`, `gssapi`, `ntlm`, `smb`, `dhcp`, `smtp`, `imap`, `ftp`, `radius`, `dce_rp`, `irc`).

service

Servizio applicativo o il protocollo utilizzato nel traffico di rete per una determinata connessione
Tipo colonna: Categorico
Valori unici: 23 - ['dns', 'http', 'ssl', 'gssapi;ntlm;smb'...]
Valori di default: 453924 (73.57%)
Valore di default: -
Moda (escluso valori di default): http



Duration:
La distribuzione dei valori risulta fortemente concentrata intorno allo zero: la maggior parte dei valori è prossima a 0, mentre alcuni valori significativamente più elevati potrebbero essere considerati outlier. In particolare, i valori superiori a 1000 si distribuiscono in modo non omogeneo tra le classi, con 10 appartenenti alla classe "normale" e 3 alla classe "DDoS".

duration

Durata della connessione o dell'attività misurata, espressa in secondi
Tipo colonna: Numerico Continuo - **Min**: 0.0, **Max**: 10946.20999, **Range**: 10946.20999
Valori di default: 0 (0.00%)
Media: 9.2816, **Moda**: 0.0, **Mediana**: 0.000406

Src_bytes e **dst_bytes**:
Le colenne presenta una distribuzione fortemente asimmetrica con la presenza di numerosi valori nulli. Sebbene la presenza di valori nulli possa essere plausibile, risulta improbabile che un numero così elevato di occorrenze sia nullo, ciò avvalora l'ipotesi dell'utilizzo del valore 0 come valore predefinito per le variabili numeriche. Analizzando src_bytes emerge inoltre un'incongruenza relativa a 19 valori stringa `0.0.0.0`, che essendo un numero ridotto sono stati eliminati per facilitare l'analisi e non alterare i dati.

src_bytes

Byte inviati dalla sorgente
Tipo colonna: Numerico Discreto
Min: 0, **Max**: 2932030659, **Range**: 2932030659
Valori di default: 0 (0.00%)
Media: 976937.3831, **Moda**: 0, **Mediana**: 0.0

dst_bytes

Byte ricevuti dal destinatario
Tipo colonna: Numerico Discreto
Min: 0, **Max**: 4291827045, **Range**: 4291827045
Valori di default: 0 (0.00%)
Media: 835144.5185, **Moda**: 0, **Mediana**: 0.0

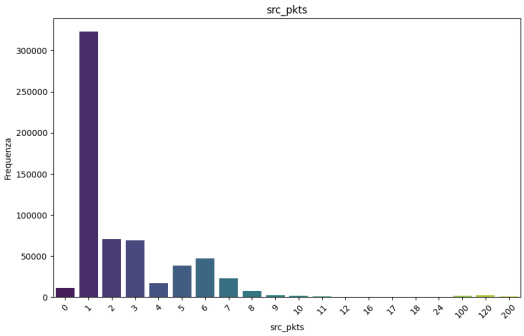
Src_pkts e **dst_pkts**:
Anche per queste colonne la distribuzione dei valori risulta asimmetrica, con una coda lunga a destra, questo suggerisce del traffico caratterizzato da piccoli scambi, con la presenza alcuni valori elevati, inoltre il valore 0 potrebbe indicare traffico nullo o particolari condizioni del traffico di rete, suggerendo possibili errori di trasmissione,o nuovamente valori di default.

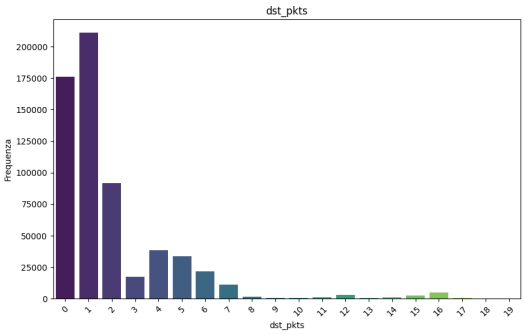
src_pkts

Pacchetti inviati dalla sorgente
Tipo colonna: Numerico Discreto
Min: 0, **Max**: 48084, **Range**: 48084
Valore di default: 11181 (1.81%)
Media: 4.3562, **Moda**: 1, **Mediana**: 1.0

dst_pkts

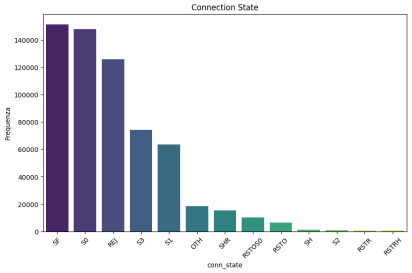
Pacchetti ricevuti dalla destinazione
Tipo colonna: Numerico Discreto
Min: 0, **Max**: 492329, **Range**: 492329
Valori di default: 0 (0.00%)
Media: 3.0923, **Moda**: 1, **Mediana**: 1.0





conn_state

Stato della connessione del traffico di rete
Tipo colonna: Categorico
Valori unici: 13 - ['OTH', 'SF', 'S0', ...]
Valori di default: 0 (0.00%)
Moda (esclusi i valori di default): SF



missed_bytes

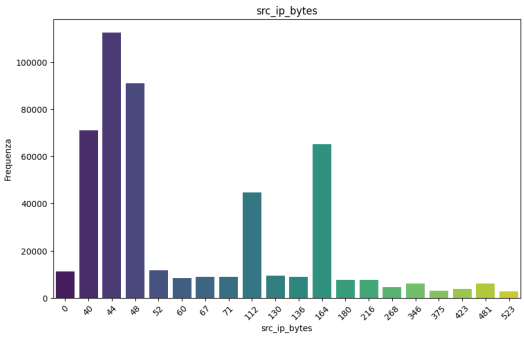
Byte persi o non ricevuti correttamente durante una trasmissione di dati
Tipo colonna: Numerico Discreto - **Min**: 0, **Max**: 2397086555, **Range**: 2397086555
Valori di default: 0 (0.00%)
Media: 84183.3632, **Moda**: 0, **Mediana**: 0.0

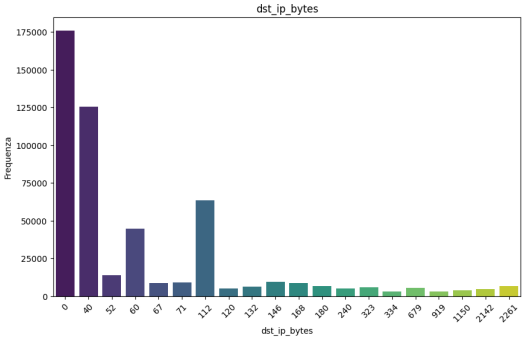
src_ip_bytes

Byte totali inviati dalla sorgente
Tipo colonna: Numerico Discreto
Min: 0, **Max**: 10418454, **Range**: 10418454
Valori di default: 0 (0.00%)
Media: 395.1811, **Moda**: 44, **Mediana**: 63.0

dst_ip_bytes

Byte totali ricevuti dalla destinazione.
Tipo colonna: Numerico Discreto
Min: 0, **Max**: 32001385, **Range**: 32001385
Valori di default: 0 (0.00%)
Media: 602.4701, **Moda**: 0, **Mediana**: 52.0





Colonne relative a DNS

Tutte le seguenti colonne presenti numerosi valori di default – o `0`, alcuni attribuibili a valori di default altri invece inseriti in quanto l'istanza non ha service dns.

dns_query

Nome di dominio interrogato durante la query DNS
Tipo colonna: Categorico,- **Valori unici**: 2231 - ['229.3.in-addr.arpa', 'ip6.arpa', ...]
Valori di default: 571606 (92.64%) - **Valore di default**: -
Moda (escluso valori di default): testphp.vulnweb.com

dns_rcode

Codice di risposta DNS, che indica il risultato della query.
Tipo colonna: Categorico,- **Valori unici**: 4 - ['0', '3', '5', '2']
Valori di default: 0 (0.00%)
Moda (escluso valori di default): 0

dns_qclass

Classe della query DNS
Tipo colonna: Categorico
Valori unici: 3 - ['0', '1', '32769']
Valori di default: 571731 - (92.66%)
Valore di default: 0
Moda (escluso valori di default): 1

dns_qtype

Tipo della query DNS
Tipo colonna: Categorico
Valori unici: 11 - ['0', '43', '48', '1', ...]
Valori di default: 571731 (92.66%)
Valore di default: 0
Moda (escluso valori di default): 1

dns_AA

Flag che indentifica la risposta da un'autorità autorizzata
Tipo colonna: Binario
Valori unici: 3 - ['F', 'T', '-']
Valori di default: 571317 (92.60%)
Valore di default: -
Moda (escluso valori di default): F

dns_RD

Flag che indentifica la ricorsione della query
Tipo colonna: Binario
Valori unici: 3 - ['F', 'T', '-']
Valori di default: 571317 (92.60%)
Valore di default: -
Moda (escluso valori di default): T

dns_rejected

Flag che indentifica se la query è rifiutata
Tipo colonna: Binario
Valori unici: 3 - ['F', 'T', '-']
Valori di default: 571317 (92.60%)
Valore di default: -
Moda (escluso valori di default): F

dns_RA

Flag che indentifica se la query può essere ricorsiva
Tipo colonna: Binario
Valori unici: 3 - ['F', 'T', '-']
Valori di default: 571317 (92.60%)
Valore di default: -
Moda (escluso valori di default): F

Colonne relative a SSL

Tutte le seguenti colonne presenti numerosi valori di default – o `0`, alcuni attribuibili a valori di default altri invece inseriti in quanto l'istanza non ha service ssl.

ssl_version

Versione del protocollo SSL utilizzato nella connessione.
Tipo colonna: Categorico, **Valori unici**: 5 - ['TLSv12', 'TLSv13', 'TLSv11', 'TLSv10', '-']
Valori di default: 616326 (99.89%) - **Valore di default**: -
Moda (escluso valori di default): TLSv10

ssl_cipher

Cifrario utilizzato per la crittografia della connessione SSL.
Tipo colonna: Categorico, **Valori unici**: 7 - ['TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256', ...]
Valori di default: 616326 (99.89%) - **Valore di default**: -
Moda (escluso valori di default): TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

ssl_resumed

Flag che indica se la connessione è stata ripresa da una precedente sessione
Tipo colonna: Binario, **Valori unici**: 3 - ['F', 'T', '-']
Valori di default: 616293 (99.89%)
Valore di default: -
Moda (escluso valori di default): F

ssl_established

Flag che indica se la connessione SSL è stata stabilita con successo.
Tipo colonna: Binario, **Valori unici**: 3 - ['F', 'T', '-']
Valori di default: 616293 (99.89%)
Valore di default: -
Moda (escluso valori di default): T

ssl_subject

Soggetto del certificato SSL
Tipo colonna: Categorico,
Valori unici: 4 - ['CN=localhost', '-', ...]
Valori di default: 616367 (99.90%)
Valore di default: -
Moda (escluso valori di default): CN=Mr Shepherd;OU=Security Shepherd Project;O=OWASP;L=Dublin;ST=Ireland;C=IE

ssl_issuer

Emmittente del certificato SSL
Tipo colonna: Categorico,
Valori unici: 4- ['CN=localhost', '-' ...]'
Valori di default: 616367 (99.90%)
Valore di default: -
Moda (escluso valori di default): CN=Mr Shepherd;OU=Security Shepherd Project;O=OWASP;L=Dublin;ST=Ireland;C=IE

Colonne relative a HTTP

Tutte le seguenti colonne presenti numerosi valori di default – o `0`, alcuni attribuibili a valori di default altri invece inseriti in quanto l'istanza non ha service http.

http_uri

URI della richiesta HTTP
Tipo colonna: Categorico
Valori unici: 58 - ['/no_nonce_string/ContentDirectory/scpd.xml', '/icon_SML.png', '/upnp/control/ConnectionManager1', '-', ...]
Valori di default: 6616265 (99.88%) - **Valore di default**: -
Moda (escluso valori di default): /

http_user_agent

Stringa identificativa del client che invia la richiesta HTTP
Tipo colonna: Categorico - **Valori unici**: 103
Valori di default: 6616265 (99.88%) - **Valore di default**: -
Moda (escluso valori di default): Mozilla/5.0 (Hydra)

http_trans_depth

Profondità delle transazioni HTTP.
Tipo colonna: Categorico - **Valori unici**: 2 - ['1', '-']
Valori di default: 6616264 (99.88%)
Valore di default: -
Moda (escluso valori di default): 1

http_status_code

Codice di stato HTTP.
Tipo colonna: Categorico
Valori unici: 8 - ['0', '200', '206', '404', '302', '400', '403']
Valori di default: 616279 - (99.88)%
Valore di default: 0
Moda (escluso valori di default): 302

http_method

Metodo HTTP utilizzato per la richiesta
Tipo colonna: Categorico
Valori unici: 3 - ['GET', 'POST', '-']
Valori di default: 6616265 (99.88%)
Valore di default: -
Moda (escluso valori di default): POST

http_version

Versione del protocollo HTTP utilizzato per la richiesta
Tipo colonna: Categorico
Valori unici: 2 - ['1.1', '-']
Valori di default: 6616279 (99.88%)
Valore di default: -
Moda (escluso valori di default): 1.1

http_request_body_len

Lunghezza del corpo della richiesta HTTP
Tipo colonna: Numerico Discreto
Valori unici: 9 - [0, 300, 40, ...]
Valori nulli: 0 (0.00%)
Media: 0.0047, **Moda**: 0, **Mediana**: 0.0
Min: 0, **Max**: 300, **Range**: 300

http_response_body_len

Lunghezza del corpo della risposta HTTP
Tipo colonna: Numerico Discreto
Valori unici: 28 - [0, 1447, 29073, ...]
Valori nulli: 0 (0.00%)
Media: 0.5782, **Moda**: 0, **Mediana**: 0.0
Min: 0, **Max**: 29073, **Range**: 29073

http_orig_mime_types

MIME type del contenuto inviato nella richiesta HTTP
Tipo colonna: Binario
Valori unici: 3 - ['application/soap+xml', 'text/plain']
Valori di default: 6616940 (99.99%)
Valore di default: -
Moda (escluso valori di default): text/plain

http_resp_mime_types

MIME type del contenuto ricevuto nella risposta HTTP
Tipo colonna: Categorico
Valori unici: 5 - ['application/xml', 'image/png', ...]
Valori di default: 6616798 (99.97%)
Valore di default: -
Moda (escluso valori di default): text/html

Colonne relative ad anomalie

weird_name

Nome evento weird (associato a particolari eventi).
Tipo colonna: Categorico, **Valori unici**: 12 - ['DNS_RR_unknown_type', 'data_before_established', ...]
Valori nulli:0, **Valori di default**: 616919 (99.99%) - **Valore di default**: -
Moda (escluso valori di default): DNS_RR_unknown_type

weird_addl

??
Tipo colonna: Categorico
Valori unici: 4 - ['46', '43', '48', '-']
Valori nulli:0, **Valori di default**: 616975 (100.00%) - **Valore di default**: -
Moda (escluso valori di default): 46

weird_notice

Flag che indica se è stato generato un avviso per un evento weird.
Tipo colonna: Binario, **Valori unici**: 2 - ['F', '-']
Valori nulli:0, **Valori di default**: 616919 (99.99%) - **Valore di default**: -,
Moda (escluso valori di default): F

Colonne target

Queste colonne sono il target scelto per la classificazione, in particolare nel progetto si è scelto di predire la classe 'type' poiché permette una classificazione multiclasse e intrinsecamente contiene 'label' in quato il tipo normal identificato identifica il traffico non malevolo.

label

Flag che indica se il traffico è o no malevolo.
Tipo colonna: Binario
Valori unici: 2 - [0, 1]
Valori nulli:0, **Valori mancanti**: 0 (0.00%)
Moda (escluso valori di default): 1

type

Tipo di attività del traffico
Tipo colonna: Categorico
Valori unici: 10 -['normal', 'scanning', 'dos', ...]
Valori nulli:0, **Valori mancanti**: 0 (0.00%)
Moda (escluso valori di default): scanning

1.2 Divisione in train e validation set

Per garantire un corretto addestramento e una reale valutazione del modello, il dataset è stato suddiviso in due insiemi distinti. Inizialmente, è stata adottata una suddivisione 80%-20% tra training set e validation set. Tuttavia, si è osservato che il modello faticava a classificare correttamente le classi con un numero ridotto di istanze. Per questo motivo, si è deciso di modificare la proporzione fino a raggiungere una suddivisione 90%-10%, al fine di fornire al modello una quantità maggiore di dati di addestramento e migliorare la capacità di generalizzazione. Nella fase di addestramento, il training set (90%) è utilizzato per ottimizzare i parametri del modello, mentre il validation set (10%) consente di valutare le prestazioni su dati non visti, fornendo un'indicazione della sua capacità di generalizzazione. Da questo punto in poi, tutte le operazioni di data cleaning e preprocessing fanno riferimento a X_train. Successivamente, le stesse trasformazioni vengono applicate a validation set, garantendo coerenza tra i due insiemi. In particolare, vengono riapplicate tutte le operazioni che prevedono una trasformazione basata sui dati, come i mapping eseguiti nel data cleaning. Invece, l'encoding delle variabili categoriali, lo scaling delle feature e la riduzione dimensionale, vengono fittate esclusivamente sul training set e poi riapplicate al validation set, evitando così il rischio di leakage tra i due insiemi.

1.3 Data Cleaning

Il processo di data cleaning ha consentito di migliorare la qualità e la coerenza dei dati, rendendoli adeguati per le analisi successive. Le operazioni di pulizia sono state effettuate seguendo l'ordine descritto di seguito.

1.3.1 Disambiguazione delle Colonne per Servizio

Una delle principali problematiche affrontate riguardava l'identificazione univoca dei valori nella colonna `service`. In alcuni casi, infatti, questa colonna conteneva più servizi associati alla stessa istanza, separati da un punto e virgola (`;`). Per risolvere questa ambiguità, è stata adottata una strategia di disambiguazione basata sulla creazione di colonne binarie per ciascun valore univoco di servizio. Ogni nuova colonna è stata denominata utilizzando il nome del servizio preceduto dal prefisso `service_`. All'interno di queste colonne, il valore `True` indica la presenza del servizio specifico nell'istanza, mentre il valore `False` ne segnala l'assenza.
Questa trasformazione ha permesso di rappresentare e identificare più servizi attivi contemporaneamente per una singola istanza. Al termine del processo, la colonna originale `service` è stata rimossa, insieme alla colonna `service_-`, utilizzata per identificare i valori di default. Inoltre, questa strategia garantisce una maggiore robustezza nei confronti di eventuali nuovi servizi non precedentemente conosciuti: in tali casi, così come per i valori di default, tutte le colonne relative ai servizi rimarranno impostate su `False`.

1.3.2 Eliminazione valori duplicati e nulli

Dalle analisi preliminari era emerso che nel dataset non fossero presenti né valori nulli né duplicati. Tuttavia, per garantire l'integrità del dataset ed evitare eventuali anomalie future, è stata implementata una procedura automatica per la loro identificazione e rimozione. Questa misura, sebbene precauzionale, assicura la qualità dei dati anche in scenari di aggiornamento o modifica del dataset originale.

1.3.3 Rimozione di Colonne

Durante l'analisi preliminare del dataset, sono state identificate diverse colonne superflue o non adatte all'addestramento dei modelli.
Un primo intervento ha riguardato l'eliminazione della colonna `ts`, poichè rappresenta un dato temporale indicante la data dell'attività e quindi non pertinenti al task. Sebbene questa variabile possa incrementare significativamente le prestazioni del modello, dai test eseguiti si raggiungono performance di 99.6%, il suo utilizzo introduce un forte rischio di overfitting. Questo accade perché il momento in cui si verifica un attacco non è intrinsecamente correlato alla tipologia di attacco, ma riflette piuttosto correlazioni spurie legate alla raccolta dei dati in un periodo di tempo ristretto. Di conseguenza, la rimozione della colonna `ts` è stata ritenuta necessaria per garantire una maggiore generalizzabilità del modello.
Un'altra colonna eliminata in via preliminare è `label`, che identifica se un'attività è malevola o meno. Questa colonna risulta infatti strettamente correlata con `type`, utilizzata come target per la classificazione. L'inclusione di `label` nel dataset avrebbe comportato un utilizzo improprio delle informazioni, compromettendo l'integrità del processo di apprendimento.

Successivamente, per modelli tradizionali come il k-Nearest Neighbors (kNN), si è proceduto all'eliminazione di colonne altamente correlate con altre. Tali colonne contengono valori ridondanti, che possono sovraccaricare il modello con informazioni superflue, limitandone la capacità di generalizzare su nuovi dati. Nel caso del kNN, un approccio più stringente nell'eliminazione delle colonne è stato considerato vantaggioso, in quanto questo modello si basa sul calcolo delle distanze tra punti nel feature space. La presenza di colonne ridondanti o con valori complessi da confrontare può alterare significativamente i risultati.

Per evitare la presenza di dati ridondanti, sono state eliminate le colonne con una correlazione molto elevata, visibile attraverso l'analisi della matrice di correlazione delle colonne categoriali, calcolata tramite Cramér's V (1.2).
Emerge una chiara correlazione tra colonne appartenenti alla stessa sottosezione del dataset, ad esempio, le variabili `ssl_version`, `ssl_resumed` e `ssl_established` presentano correlazioni rispettivamente pari a 1, 0.79 e 0.75 con `ssl_cipher`, rendendole superflue. Analogamente, sono state rimosse `dns_qclass`, `dns_qtype`, `dns_rcode`, `dns_RD`, `dns_AA` e `dns_rejected`, in quanto altamente correlate con `dns_RA`. Per quanto riguarda le variabili relative al protocollo HTTP, le colonne `http_method`, `http_version`, `http_orig_mime_types`, `http_resp_mime_types`, `http_request_body_len` e `http_response_body_len` sono risultate fortemente correlate con `http_trans_depth` e `http_status_code`, e pertanto sono state rimosse.

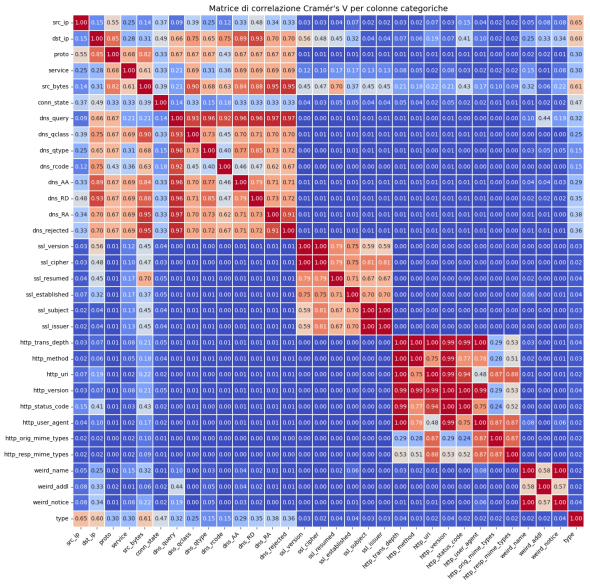


Fig. 1.2: Matrice di correlazione delle colonne categoriche

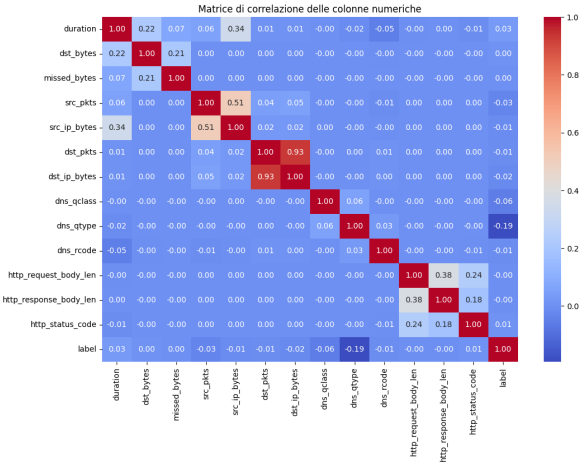


Fig. 1.3: Matrice di correlazione delle colonne numeriche

Infine, le variabili `weird_addl` e `weird_notice`, che mostrano un'alta correlazione con `name`, sono state escluse per ridurre la ridondanza informativa.
Questa selezione ha permesso di eliminare informazioni duplicate, mantenendo le colonne maggiormente correlate a `type` e alle altre variabili del rispettivo sottogruppo, garantendo la presenza di una o al massimo due colonne per ciascun sottogruppo.

Al contrario, per modelli basati su reti neurali, come le reti Feed Forward e TabTransformer, si è preferito mantenere un numero maggiore di colonne, incluse quelle correlate. Infatti grazie alla loro architettura avanzata, questi modelli sono in grado di apprendere relazioni latenti tra le feature, mitigando il rischio di overfitting associato alla presenza di colonne correlate. Questa scelta ha permesso di sfruttare appieno la capacità delle reti neurali di modellare interazioni complesse tra le variabili, migliorando la qualità delle previsioni senza compromettere la generalizzabilità.

1.3.4 Mapping e Casting delle Colonne

Un ulteriore problema emerso durante l'analisi preliminare è legato all'identificazione non sempre corretta delle colonne: in particolare, alcune colonne classificate come numeriche discrete risultavano invece essere di natura categorica, come ad esempio la colonna che rappresenta il numero delle porte.
In primo luogo, si è ritenuto necessario implementare una mappatura esplicita per migliorare la leggibilità e l'interpretazione dei dati, oltre a ridurre il numero di valori discreti rappresentati nelle colonne.
A tal fine, sono state effettuate le seguenti operazioni:

- **Indirizzi IP**: Le colonne `src_ip` e `dst_ip` sono state categorizzate in gruppi come "Private", "Public", "Loopback", ecc., utilizzando una libreria specializzata per l'elaborazione degli indirizzi IP, al fine di fornire una classificazione più chiara e funzionale.
- **Porte di Comunicazione**: Le colonne relative alle porte (`src_port`, `dst_port`) sono state suddivise in tre principali intervalli:
  - *Well-Known* (porte da 0 a 1023);

  - *Registered* (porte da 1024 a 49151);
  - *Dynamic* (porte da 49152 a 65535).

  Questo approccio consente di distinguere con maggiore precisione il tipo di traffico rappresentato.

- **Risposte DNS**: La colonna `dns_rcode` è stata rimappata in categorie più descrittive, come `'No Error'`, `'ServerFailure'`, `'NameError'` e `'Refuse'`, per agevolare la comprensione delle risposte del protocollo DNS.
- **Colonne Categoriali**: Colonne come `dns_qtype` e `dns_qclass` sono state rimappate in stringhe che identificano chiaramente il tipo e la classe della query rappresentata, rendendo i dati più interpretabili.
- **Valori Booleani**: Le colonne booleane contenenti valori rappresentati da caratteri (`T`/`F`), sono state trasformate in valori booleani (`True`/`False`). Successivamente, è stato eseguito un cast a string per uniformare il formato e rendere le colonne categoriche.

Una volta completata la fase di mappatura, le colonne erroneamente classificate come numeriche discrete, come `weird_addl` e `http_status_code`, sono state convertite in stringhe per garantire un'identificazione corretta.

1.3.5 Discrepancy Detection

L'ultima problematica riscontrata durante l'analisi dei dati riguardava l'ambiguità dei valori di default nel distinguere tra valori che rappresentano un'impostazione predefinita e valori che, indicano l'assenza di dati. Ad esempio se è attivo protocollo HTTP, le colonne relative ai protocolli DNS e SSL dovrebbero presentare valori nulli non predefiniti per non introdurre confusione nell'interpretazione e nell'elaborazione dei dati.
Si è quindi distinto i casi che presentavano valori di assenza ,sostituendo con un simbolo (`/`) i valori delle colonne corrispondenti ai servizi non attivi.
In questo modo, si è ridotto il rischio di errori interpretativi e si è garantita una maggiore chiarezza nell'utilizzo delle informazioni durante le fasi di analisi ed elaborazione del modello.

1.3.6 Sostituzione dei valori di default

Sebbene il dataset non contenga valori nulli, sono presenti numerosi valori di default che contribuiscono ad un'elevata sparsità di dati. Per affrontare questa problematica, è stata adottata una strategia di sostituzione di tali valori con misure di centralità, al fine di migliorare le prestazioni durante l'addestramento del modello.
Una delle principali difficoltà emerse riguarda il valore zero (0), che potrebbe non essere identificabile come valore di default, in quanto potrebbe rappresentare un dato valido. Tuttavia, la sua elevata frequenza suggerisce che, per alcune variabili numeriche, sia stato effettivamente utilizzato come valore predefinito.
Sono state testate diverse strategie di sostituzione:

- Sostituzione del valore di default '-' con la moda della variabile.
- Sostituzione dei valori di default '-' e '/' (indicante l'assenza del servizio selezionato) con la moda

1.3.7 Gestione degli Outlier

L'ultimo passaggio del processo di data cleaning ha riguardato l'individuazione e la rimozione degli outlier, al fine di evitare che influenzassero negativamente le prestazioni del modello.
Sono stati implementati diversi metodi per l'identificazione e l'eliminazione degli outlier, scartando quelli che hanno prodotto risultati inferiori, come il metodo IQR (Interquartile Range) e la tecnica Limited Base, caratterizzata da soglie meno restrittive rispetto all'approccio standard. Inoltre, sono state valutate e scartate alcune metodologie che combinano più tecniche di base. Al termine dell'analisi, sono stati selezionati i metodi che hanno garantito le migliori performance.

- **No**: Nessun trattamento sugli outlier, utilizzando i dati nella loro forma originale.
- **Base (soglie fisse)**: Applicazione di valori limite predefiniti per specifiche feature, determinati a seguito di un'analisi preliminare del dataset, con l'obiettivo di eliminare valori anomali evidenti.
- **Percentile**: Rimozione delle osservazioni estreme sulla base di soglie percentile (il 1° e il 99° percentile). Questo metodo è stato scelto per la sua maggiore robustezza rispetto a distribuzioni non normali, come quelle tipicamente presenti nel dataset analizzato.
- **Isolation Forest**: Approccio di apprendimento non supervisionato, selezionato per la sua capacità di distinguere tra punti normali e outlier anche in situazioni in cui tale separazione non è immediatamente evidente analizzando spazi multidimensionali.
- **Dynamic Threshold**: Applicazione di soglie dinamiche basate sulla media e sulla deviazione standard per determinare un intervallo di valori considerati accettabili. Questa tecnica ha mostrato buoni risultati nelle fasi preliminari, sebbene si basi sull'assunzione che i dati abbiano una distribuzione relativamente simmetrica e che la maggior parte dei valori sia concentrata attorno alla media, ipotesi che non sempre si verifica nel dataset analizzato.

Per migliorare ulteriormente l'accuratezza della rimozione degli outlier, nei metodi **Isolation Forest**, **Percentili** e **Dynamic Threshold** l'analisi degli outlier è stata condotta separatamente per ciascuna classe, anziché sull'intero dataset. Questo approccio ha permesso una rilevazione e rimozione più efficace, garantendo una maggiore coerenza nella distribuzione dei dati e migliorando le prestazioni complessive del modello.

1.4 Data Preprocessing

La fase di preprocessing ha l'obiettivo di minimizzare l'errore riducibile della funzione predittiva, migliorando così l'efficacia del modello. In questo contesto, sono state valutate diverse tecniche di preprocessing, tra cui la gestione degli outlier, la scalatura delle feature, l'encoding delle variabili categoriali, il bilanciamento delle classi, al fine di selezionare l'approccio più adeguato per ogni modello considerato.
Le operazioni di preprocessing sono state effettuate seguendo l'ordine descritto di seguito.

1.4.1 Encoding delle Variabili Categoriali

Nel dataset erano presenti diverse variabili categoriali , per convertire queste variabili in un formato compatibile con gli algoritmi di apprendimento automatico, è stato adottato l'One-Hot Encoding. Questo metodo consente di trasformare ciascuna categoria in un vettore binario, evitando di introdurre un ordine arbitrario tra le classi.
Il One-Hot Encoding genera una nuova colonna per ogni categoria distinta della variabile, assegnando il valore 1 se l'osservazione appartiene a quella categoria e 0 altrimenti. Sebbene questa tecnica possa aumentare significativamente la dimensionalità del dataset, è stata preferita poiché garantisce che il modello non attribuisca relazioni spurie tra categorie non numericamente comparabili.
Per preservare la coerenza tra il training set e i set di validazione/test, l'encoder è stato inizialmente calcolato esclusivamente sui dati di addestramento. Successivamente, è stato salvato e riutilizzato per trasformare i dati di validazione evitando discrepanze dovute a categorie eventualmente non presenti nei sottoinsiemi di dati futuri.

1.4.2 Bilanciamento delle Classi

Il bilanciamento delle classi è stato un aspetto cruciale del nostro lavoro, infatti considerando la differenza di dimensionalità delle classi, in cui la classe maggioritaria contava un valore superiore a 200.000 istanze, mentre quella minoritaria poco più di 500. Per affrontare questa problematica e garantire un'equa rappresentazione delle classi durante l'addestramento del modello, sono stati testati diversi livelli di bilanciamento, uniformando il numero di istanze a 15.000, 20.000 e 25.000 valori per classe.

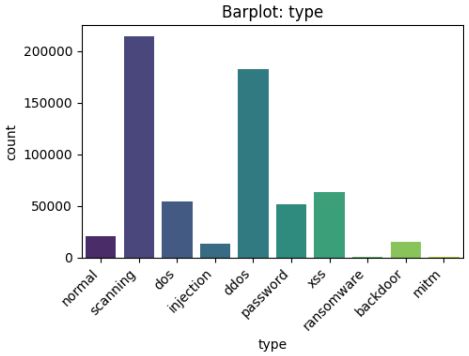


Fig. 1.4: Distribuzione delle istanze per classe

Tre diverse tecniche di bilanciamento sono state sperimentate:

- le istanze delle classi meno rappresentate sono state replicate fino a raggiungere il valore target, mentre quelle delle classi maggioritarie sono state ridotte tramite undersampling.
- l'algoritmo SMOTE è stato utilizzato per generare nuovi dati sintetici per le classi minoritarie, abbinato a undersampling per ridurre le istanze delle classi maggioritarie.
- è stato applicato SMOTE solo alle classi con un numero di istanze inferiore alla metà del target desiderato, mentre per le classi con più della metà delle istanze si è optato per la ripetizione. L'undersampling è stato invece utilizzato per ridurre le classi con un numero di istanze superiore al target.

Tra le tre tecniche, la terza si è dimostrata la più efficace, fornendo i migliori risultati durante i test. Di conseguenza, è stata adottata come approccio finale, mentre le altre due sono state scartate. Per garantire robustezza e verificare l'impatto della casualità nelle operazioni di undersampling, sono stati generati cinque dataset distinti, variando il seed randomico utilizzato per questa fase.

Per i modelli basati su reti neurali e TabTransformer, l'approccio standard inizialmente non ha prodotto risultati soddisfacenti. Un'analisi delle performance per classe ha evidenziato difficoltà nella predizione delle classi 4 e 7, caratterizzate da una scarsa rappresentatività nei dati. Per migliorare le prestazioni, sono state quindi esplorate tecniche alternative a SMOTE.

- ADASYN (Adaptive Synthetic Sampling): testata come alternativa a SMOTE, ha però mostrato risultati inferiori.
- Generative Adversarial Networks (GANs): sebbene promettenti, si sono rivelate troppo onerose in termini di tempo computazionale.
- BorderlineSMOTE: individuata come la variante di SMOTE più efficace, ha migliorato la capacità del modello di apprendere rappresentazioni più significative delle classi minoritarie.

Infine, per i modelli di deep learning su dati tabulari, si è anche testato un approccio basato sulla pesatura delle classi anziché sull'oversampling, ma sebbene questa tecnica producesse risultati comparabili, è stata scartata in favore del bilanciamento diretto, poiché permetteva una significativa riduzione dei tempi di computazione.

1.4.3 Scalatura e Normalizzazione

L'utilizzo di tecniche di scalatura e normalizzazione è cruciale nell'analisi dei dati, soprattutto quando il dataset contiene variabili con distribuzioni molto diverse tra loro. In queste situazioni, le differenze di scala possono influire negativamente sulla performance dei modelli, in particolare quelli che si basano su metriche di distanza, come le SVM, KNN o le reti neurali. Variabili con scale più estese possono dominare il processo di ottimizzazione, rendendo difficile per il modello apprendere relazioni significative tra le feature.
Le tecniche di scalatura e normalizzazione aiutano a uniformare i valori delle feature, garantendo che ogni variabile contribuisca in modo proporzionato al modello. Di seguito vengono descritti i metodi principali analizzati e utilizzati nel progetto.

- **StandardScaler**: Trasforma i dati in una distribuzione con media $\mu = 0$ e deviazione standard $\sigma = 1$. Questa tecnica è indicata per feature normalmente distribuite. La formula utilizzata è:

$$x' = \frac{x - \mu}{\sigma} \tag{1.1}$$

  dove μ e σ rappresentano, rispettivamente, la media e la deviazione standard della feature.

- **MinMaxScaler**: Restringe i valori delle feature in un intervallo predefinito, tipicamente $[0, 1]$. Questo è utile per feature con range ben definiti. La formula utilizzata è:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{1.2}$$

  dove $x_{\min}$ e $x_{\max}$ sono il valore minimo e massimo della feature, rispettivamente.

- **QuantileTransformer**: Trasforma la distribuzione delle feature in una distribuzione uniforme o normale. La trasformazione è basata sulla stima dei quantili. Questo metodo è utile per ridurre l'asimmetria o rendere i dati più aderenti alle ipotesi del modello.

- **L1 Normalization (Manhattan Norm)**: Scala ogni valore della feature dividendo per la somma assoluta dei valori della stessa feature. Questa tecnica è utile quando si vuole dare lo stesso peso a tutte le feature indipendentemente dalla loro magnitudine ed è particolarmente efficace in presenza di dati sparsi. La formula utilizzata è:
$$x' = \frac{x}{\sum |x|} \tag{1.3}$$
dove il denominatore rappresenta la somma dei valori assoluti della feature.

- **L2 Normalization (Euclidean Norm)**: Scala ogni valore della feature dividendo per la norma euclidea (o norma L2). Questo metodo è utile per garantire che le feature abbiano una lunghezza unitaria nello spazio vettoriale, spesso utilizzato nei modelli basati sulla distanza. La formula è:
$$x' = \frac{x}{\sqrt{\sum x^2}} \tag{1.4}$$
dove il denominatore rappresenta la radice quadrata della somma dei quadrati dei valori della feature.

1.4.4 Riduzione della Dimensionalità

Per affrontare la complessità e l'alta dimensionalità dei dati, sono state esplorate diverse tecniche di riduzione della dimensionalità. Questi metodi non solo aiutano a migliorare l'efficienza computazionale, ma possono anche aumentare la capacità del modello di generalizzare, riducendo il rischio di overfitting.

- **Principal Component Analysis (PCA)**: Questa tecnica non supervisionata proietta i dati in un nuovo spazio con meno dimensioni, preservando la varianza massima. La PCA è stato utilizzata per identificare le componenti principali che catturano la maggior parte dell'informazione presente nel dataset.
- **Linear Discriminant Analysis (LDA)**: Essendo una tecnica supervisionata, l'LDA riduce la dimensionalità massimizzando la separazione tra le classi.
- **Feature Importance con Random Forest (RF)**: Per i modelli basati su alberi, come le Random Forest, è stata sfruttata l'importanza delle feature calcolata in base alla riduzione dell'impurità. Questo approccio ha permesso di identificare e selezionare le variabili più rilevanti per il problema in esame.

1.5 Modeling

Per affrontare il problema della classificazione multiclasse degli attacchi, sono stati testati diversi modelli di machine learning e deep learning, scelti per la loro capacità di gestire dati con caratteristiche eterogenee e distribuzioni complesse. I modelli tradizionali utilizzati includono Support Vector Machines (SVM), Random Forest (RF) e k-Nearest

Neighbors (k-NN), noti per la loro robustezza ed efficacia in scenari strutturati. Questi sono stati affiancati da approcci avanzati basati su deep learning, tra cui reti neurali profonde (DNN), TabNet e TabTransformer, selezionati per la loro capacità di catturare relazioni complesse e adattarsi efficacemente ai dati tabulari.

Capitolo 2

Implementazione

2.1 Machine Learning Supervised tradizionali

2.1.1 SVM

Implementazione del Support Vector Machine (SVM)

L'algoritmo *Support Vector Machine* (SVM) è stato implementato per la classificazione multiclasse, sfruttando la sua capacità di separare le classi massimizzando il margine decisionale. Il modello è stato implementato tramite la libreria `scikit-learn`.

Preparazione del dataset La preparazione dei dati è stata condotta seguendo i passaggi descritti nei paragrafi 1.3 e 1.4. In particolare, è stata adottata una strategia di pulizia rigorosa, eliminando le colonne fortemente correlate e mantenendo esclusivamente quelle di maggiore rilevanza per il modello. Inoltre, si è scelto di rimuovere le variabili categoriali con un numero elevato di valori distinti al fine di evitare un'eccessiva crescita della dimensionalità del dataset.
Per eliminare ambiguità la colonna `service` è stata trasformata in un set di colonne binarie, una per ciascun valore unico, ponendo a True le colonne corrispondenti ai servizi attivi. Per migliorare la coerenza e l'interpretabilità dei dati, gli indirizzi IP sono stati categorizzati in classi, mentre le porte di comunicazione sono state suddivise nei tre intervalli standard: `Well-Known, Registered` e `Dynamic`, infine le variabili booleane sono state uniformate a valori con la sostisuzione dei carattere `T/F` con `True/False`.
Inoltre, per le colonne categoriche in cui era possibile identificare univocamente i valori di default sono state applicate tecniche di sostituzione di valori di default con la moda della colonna.
Poiché le Support Vector Machines (SVM) sono particolarmente sensibili alla scala delle feature, sono stati applicati diversi metodi di normalizzazione e scaling, tra cui *Standard Scaling*, *MinMax Scaling* e *Quantile Transformation*. Per mitigare l'influenza degli outlier, si è scelto di addestrare modelli variando la strategia di rimozione degli outlier. Infine, è stato provato l'addestramento del modello sia con che senza tecniche di riduzione della dimensionalità, in particolare sono state testate tecniche di *Principal Component Analysis* (PCA) e *Linear Discriminant Analysis* (LDA),

Addestramento e Tuning degli Iperparametri

Tipo di Iperparametro	Valori Testati
dataset	dataset_1, dataset_2, dataset_3, dataset_4, dataset_5
target count	15000
scaling methods	minmax, quantile, standard
Outlier methods	no, base, percentile, dynamic_threshold, isolation_forest
dimensionality reduction	no, LDA, PCA
PCA threshold	0.99
kernel	linear, poly
C	0.1, 0.2, 0.5
gamma	0.1, 0.2, 0.5
degree	3, 4, 5
replace_value	no, mode

Tab. 2.1: Iperparametri testati

L'addestramento è stato eseguito utilizzando cinque dataset distinti ottenuti dalla fase di bilanciamento, con l'obiettivo di valutare la robustezza del modello su configurazioni differenti. La fase di tuning degli iperparametri è stata implementata tramite grid search, esplorando le seguenti configurazioni:

- **Kernel**: Sono stati testati sia il kernel `linear` che il kernel `polynomial`. Il kernel lineare è utile per problemi lineari, mentre il kernel polinomiale consente di catturare relazioni più complesse nei dati.
- **Parametro di regolarizzazione (C)**: sono stati esplorati i valori di {0.1, 0.2, 0.5}. Il parametro C controlla il trade-off tra massimizzare il margine e minimizzare l'errore di classificazione.
- **Gamma**:sono stati utilizzati valori di {0.5, 0.1, 0.2}. Gamma definisce l'influenza di un singolo esempio di addestramento. Valori alti di gamma significano che ogni esempio ha una stretta influenza, mentre valori bassi significano che l'influenza è più ampia.
- **Grado del kernel polinomiale (degree)**: testati i valori $\{3, 4, 5\}$ per valutare l'impatto della complessità polinomiale.

Il modello è stato valutato utilizzando le seguenti metriche (calcolate sia sull'intero modello che sulle singole classi):

- **Accuratezza**: per misurare la percentuale complessiva di predizioni corrette.
- **F1-score**: utile in presenza di classi sbilanciate, combinando precision e recall.
- **Precision e Recall**: per valutare rispettivamente la qualità delle predizioni positive e la capacità di identificare correttamente le istanze positive.

Risultati L'analisi delle prestazioni del modello SVM in funzione delle diverse configurazioni iperparametriche ha evidenziato pattern significativi.
In particolare, il confronto tra i metodi di scaling, riportato nel box plot (Figura 2.1), mostra che l'utilizzo del *Quantile Transformer* porta a risultati nettamente superiori rispetto a *MinMaxScaler* e *StandardScaler*. Le statistiche descrittive confermano questa tendenza: il primo quartile (*Q1*) per il metodo quantile è pari a 0.8725, con una mediana di 0.9109 e un massimo di 0.9738, contro una mediana di 0.7423 per *MinMaxScaler* e di 0.6275 per *StandardScaler*. Quest'ultimo, in particolare, ha mostrato una maggiore variabilità e risultati meno stabili, con una minima performance estremamente bassa (0.0033).

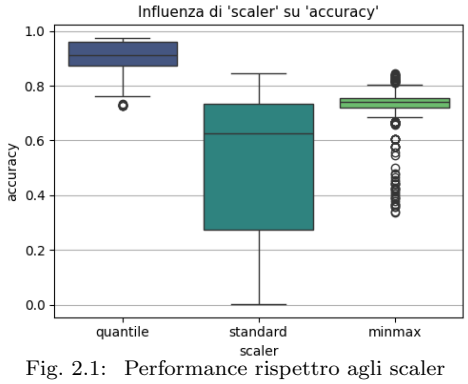


Fig. 2.1: Performance rispettro agli scaler

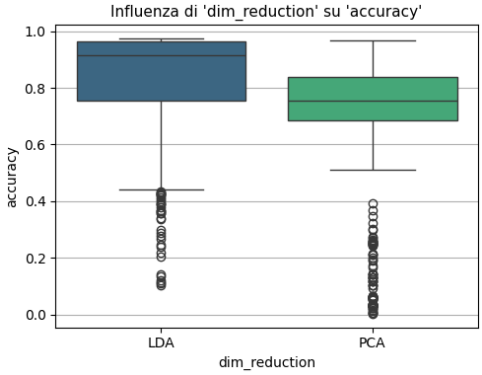


Fig. 2.2: Performance rispetto a PCA e LDA

Un ulteriore elemento emerso dall'analisi riguarda l'impatto della riduzione della dimensionalità 2.2. L'uso di *LDA* ha dimostrato di migliorare significativamente le prestazioni rispetto a *PCA*. Dopo test preliminari, la soglia della varianza cumulativa per *PCA* è stata fissata al 99%, in quanto valori inferiori hanno portato a un drastico deterioramento delle prestazioni del modello.

Analizzando l'impatto degli outlier emerge che le migliori performance si ottengono senza effettuare alcuna rimozione o eliminando solamente gli outlier più estremi, come mostrato nella tabella 2.3.

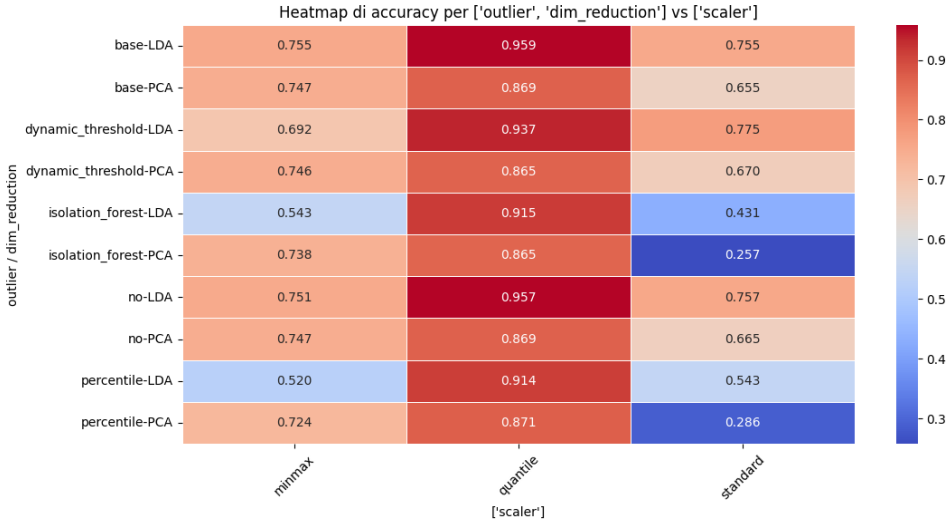


Fig. 2.3: Accuratezza media raggruppata per outlier/dim_reduction vs metodo di scaling/normalizzazione

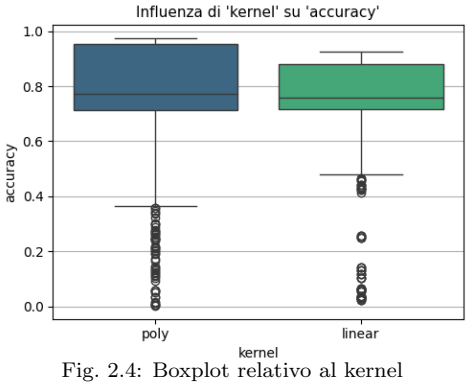


Fig. 2.4: Boxplot relativo al kernel

Per quanto riguarda la scelta del kernel, i risultati indicano una chiara superiorità della funzione polinomiale rispetto a quella lineare. Il modello con kernel polinomiale ha raggiunto una mediana di 0.7739 e un massimo di 0.9738, mentre il kernel lineare ha mostrato una maggiore dispersione delle performance, con una mediana di 0.7571 e un massimo di 0.9251. Questo risultato suggerisce che la trasformazione non lineare dei dati operata dal kernel polinomiale consente una migliore separabilità delle classi, facilitando l'apprendimento del modello. Infine, il tuning dei parametri `C`, `degree` e `degree` ha evidenziato una tendenza al miglioramento delle prestazioni all'aumentare dei loro valori (2.7). Tuttavia, limiti computazionali hanno impedito l'esplorazione di valori di `C` più elevati, poiché il modello non terminava l'addestramento neanche con la riduzione della dimensionalità applicata.

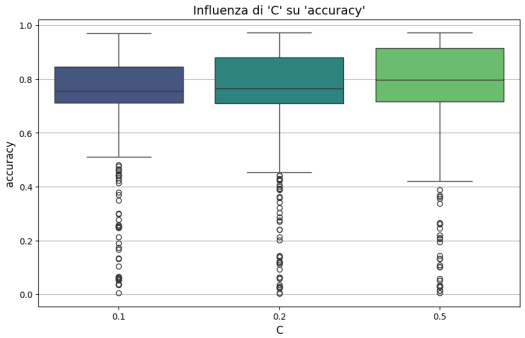


Fig. 2.5: Performance rispetto al C

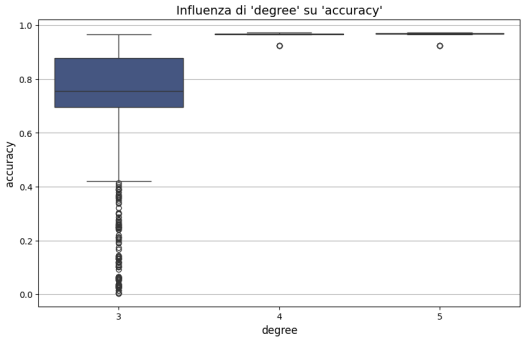


Fig. 2.6: Performance rispetto al degree

Questo suggerisce che, pur essendo possibile ottenere risultati ancora migliori con un tuning più approfondito, il costo computazionale rappresenta un vincolo significativo nell'implementazione pratica del modello.
L'esplorazione ha evidenziato che il kernel `polynomial` con grado 5 e parametri ottimizzati (`C` = 0.5, `gamma` = 0.5) ha prodotto le migliori prestazioni. Per quanto riguarda i metodi di scaling, il `QuantileScaler` si è rivelato più efficace per garantire una convergenza stabile del modello. Infine, l'utilizzo dei LDA ha contribuito a bilanciare efficienza computazionale e accuratezza predittiva.

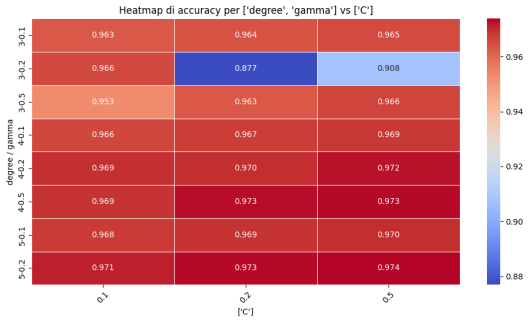


Fig. 2.7: Valore massimo dell'accuratezza raggruppata per kernel vs gamma/degree

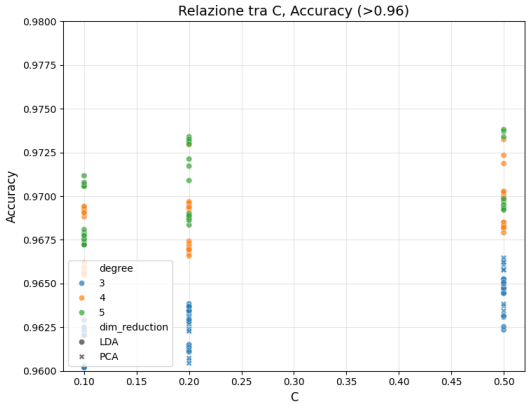


Fig. 2.8: Grafico della performance con kernel polinomiale in funzione del parametro C

2.1.2 Random Forest

Implementazione e Addestramento del Random Forest

L'algoritmo *Random Forest* è stato implementato utilizzando la libreria `scikit-learn`, con l'obiettivo di valutare le sue prestazioni nella classificazione multiclasse. Data la sua natura basata su un insieme di alberi decisionali, il modello è stato testato su diverse configurazioni iperparametriche, esplorando l'impatto di vari metodi di preprocessamento per migliorare la generalizzazione e la stabilità dei risultati.

Preparazione del dataset Il preprocessing per il Random Forest ha seguito un approccio simile a quello adottato per SVM, seguendo i passaggi descritti nei paragrafi 1.3 e 1.4. Anche in questo caso, le colonne altamente correlate sono state rimosse per evitare ridondanze e ridurre il rischio di overfitting e le variabili categoriche con un numero eccessivo di valori distinti sono state eliminate. Si è eseguita la disambiguazione della colonna `service` creando una colonna per ogni servizio secondo e ponendo a True i valori corrispondenti al servizio.
Per migliorare la coerenza e l'interpretabilità dei dati, gli indirizzi IP sono stati categorizzati in classi, le porte di comunicazione sono state suddivise nei tre intervalli e le variabili booleane sono state uniformate a valori True/False.
Il modello è stato testo sia in presenza che senza diverse tecniche di sostituzione dei valori nulli, sostituendo per le colonne categorica la media della colonna al valore di default –. Anche in questo caso sono stati testati diverse tecniche di outlier e scaling. Inoltre, per migliorare le performance del modello e ridurre la complessità computazionale, sono state testate entrambe le tecniche di riduzione della dimensionalità (*PCA, LDA*).

Addestramento e Tuning degli Iperparametri

Tipo di Iperparametro	Valori Testati
dataset	dataset_1, dataset_2, dataset_3, dataset_4, dataset_5
target count	15000
scaling methods	standard, quantile, minmax, l1, l2
outlier methods	no, isolation_forest, base, percentile, dynamic_threshold
dimensionality reduction	PCA, LDA
n_estimators	100, 200, 300, 500, 700
max_depth	5, 10, 15, 20
criterion	gini, entropy, log_loss
ccp_alpha	0.0, 0.0005, 0.001, 0.005
bootstrap	0, 1
replace_value	no, mode

Tab. 2.2: Iperparametri testati

L'addestramento è stato condotto su cinque dataset bilanciati, derivanti dal preprocessing, per garantire una valutazione equa delle prestazioni del modello. È stata effettuata un'analisi iperparametrica approfondita mediante grid search, testando diverse configurazioni:

- **Numero di alberi** (`n_estimators`): $\{100, 200, 300, 500, 700\}$.
- **Profondità massima** (`max_depth`): $\{5, 10, 15, 20\}$.

- **Criterio di suddivisione**: *Gini*, *Entropy*, *Log Loss*.
- **Pruning** (`ccp_alpha`): $\{0.0, 0.0005, 0.001, 0.005\}$.
- **Campionamento con sostituzione** (`bootstrap`): $\{0, 1\}$.

Risultati L'analisi delle prestazioni del modello Random Forest ha evidenziato l'impatto significativo di specifiche configurazioni iperparametriche sul suo comportamento.

In primo luogo, la gestione degli outlier ha avuto un'influenza determinante sulle performance (Figura 2.9). Le strategie basate su *Isolation Forest*, *Dynamic Threshold* e *Percentile* hanno portato a una riduzione delle prestazioni, risultando meno efficaci rispetto all'assenza di rimozione degli outlier. Per questo motivo, tali approcci sono stati scartati ed eliminati dalle analisi successive.

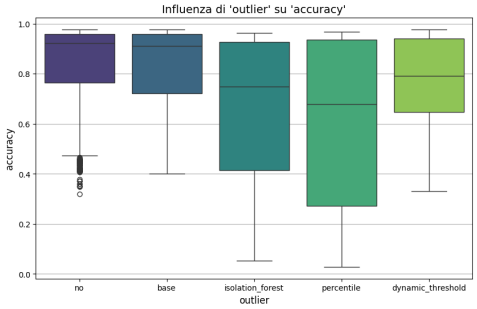


Fig. 2.9: Performance rispetto agli outlier

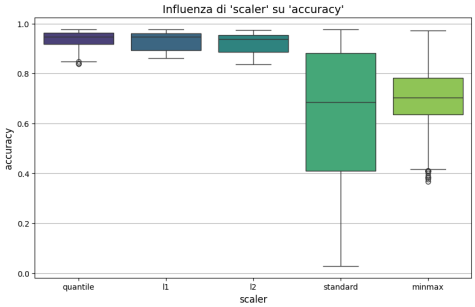


Fig. 2.10: Performance rispetto agli scaler

L'analisi dei metodi di scaling ha mostrato differenze marcate tra le tecniche valutate. I risultati indicano chiaramente che *MinMaxScaler* e *StandardScaler* hanno fornito performance inferiori rispetto agli altri metodi di normalizzazione. In particolare, il primo quartile (*Q1*) e la mediana per entrambi risultano sensibilmente più bassi rispetto a *L1*, *L2* e *Quantile Transformer*, che invece hanno garantito risultati più stabili e performanti.

A partire da questo punto, l'analisi delle prestazioni è stata filtrata considerando esclusivamente le configurazioni di outlier e scaling che hanno mostrato risultati soddisfacenti.

L'analisi delle tecniche di riduzione della dimensionalità conferma che *LDA* porta a risultati migliori rispetto a *PCA*, come evidenziato anche dal boxplot delle due configurazioni (Figura 2.12). Inoltre, il confronto tra i criteri di suddivisione (`gini`, `entropy` e `log_loss`) evidenzia che i migliori risultati si ottengono con `log_loss` ed `entropy`, le cui distribuzioni di accuratezza risultano molto simili e superiori rispetto a `gini` (Figura 2.11).

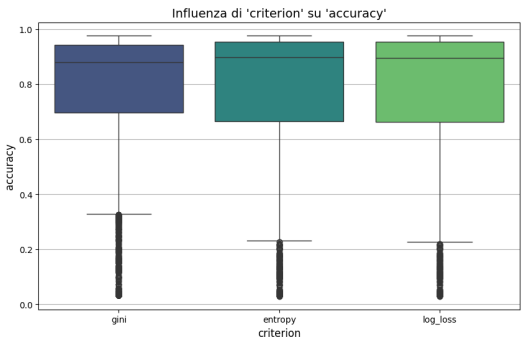


Fig. 2.11: Boxplot delle performance rispetto al criterio di split

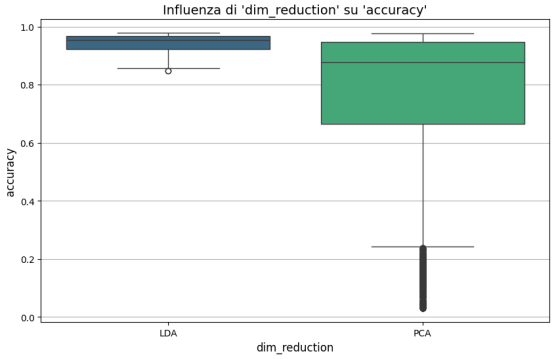


Fig. 2.12: Boxplot delle performance rispetto alla riduzione dimensionale

L'effetto dell'iperparametro `ccp_alpha` è stato studiato attraverso il grafico della sua relazione con l'accuratezza 2.13. L'analisi mostra una chiara tendenza alla diminuzione delle prestazioni all'aumentare del valore di `ccp_alpha`, suggerendo che una regolarizzazione troppo intensa penalizza la capacità del modello di apprendere dai dati.

Un altro fattore chiave nelle prestazioni del modello è la profondità massima degli alberi. Dall'analisi della relazione tra `max_depth` e accuratezza (Figura 2.15), emerge che la configurazione ottimale è una profondità pari a 15. La relazione tra il parametro `max_depth` e `max_criterion` è resa esplicita anche dalla tabella 2.14 che mostra la media dell'accuratezza per ciascun raggrappamento.

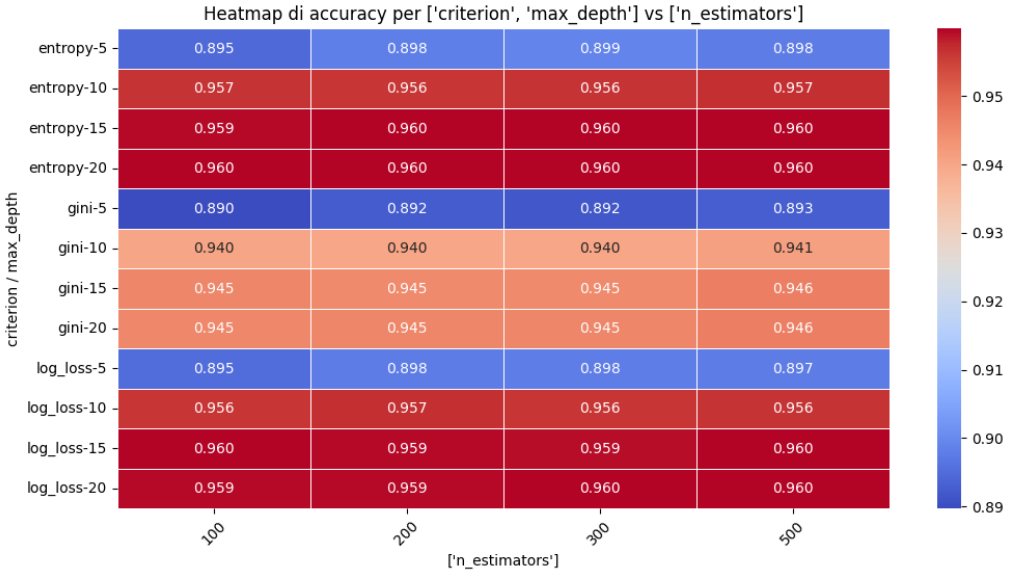


Fig. 2.14: 0
Accuratezza media raggruppata per criterion/max_depth vs n_estimators

Anche il parametro `bootstrap` influisce sulle prestazioni: impostarlo a *False* porta a un miglioramento rispetto alla configurazione con *True*, suggerendo che la costruzione degli alberi su tutti i dati disponibili piuttosto che su campioni casuali consente una migliore generalizzazione (2.16). Infine, il numero di stimatori ha mostrato un impatto significativo sulle prestazioni del modello (2.17). Dall'analisi della relazione tra il numero di alberi e l'accuratezza, emerge che il miglior compromesso tra complessità computazionale e performance si ottiene con 100 o 300 stimatori.

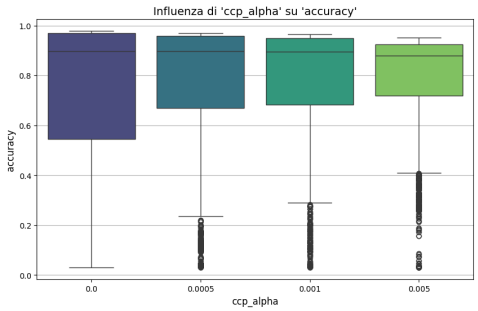


Fig. 2.13: Performance rispetto a ccp_alpha

In conclusione, il tuning del modello Random Forest ha permesso di identificare una configurazione ottimale caratterizzata dall'uso di *L1* per la normalizzazione, l'assenza di tecniche di rimozione degli outlier, il criterio `log_loss` o `entropy`, una profondità massima pari a 15 o 20, e un numero di alberi pari a 100 o 300. Queste scelte hanno garantito un equilibrio tra accuratezza e stabilità del modello, migliorando la sua capacità di generalizzazione.

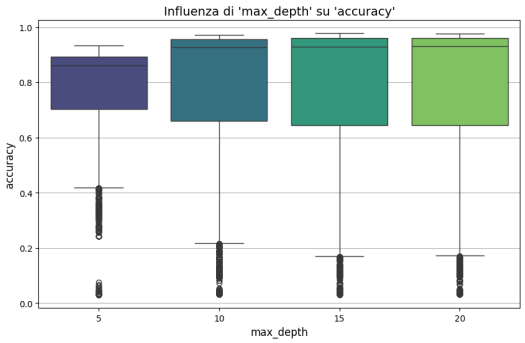


Fig. 2.15: Performance rispetto a max depth

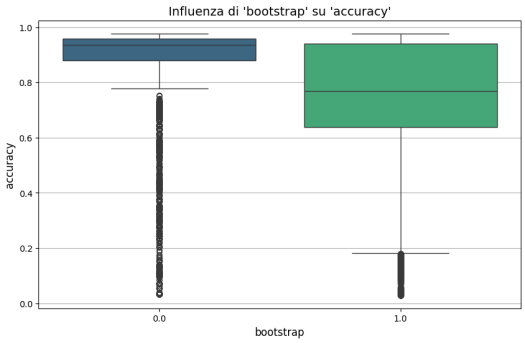


Fig. 2.16: Performance rispetto a bootstrap

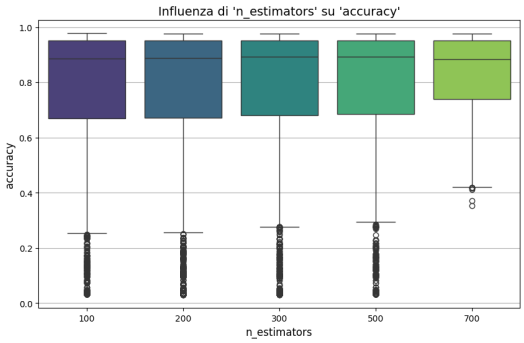


Fig. 2.17: Boxplot delle performance rispetto al numero di stimatori

Feature Importance

Dopo aver sperimentato una riduzione della dimensionalità utilizzando la Linear Discriminant Analysis (LDA), che ha mostrato buoni risultati nel migliorare la separabilità tra le classi e ridurre il rumore nei dati. Tuttavia, dato il costo computazionale della trasformazione e la necessità di mantenere l'interpretabilità delle feature originali, si è successivamente esplorato un approccio alternativo basato sulla feature importance di Random Forest.

Si è scelto di selezionare solo le feature con i punteggi di importanza più elevati, eliminando quelle con un contributo minimo. Tuttavia, questa riduzione non ha portato ai miglioramenti sperati: il modello ha subito un calo significativo delle performance. Questo effetto negativo è stato attribuito alla perdita di informazioni discriminanti tra classi meno rappresentate, come MITM e ransomware, che, sebbene abbiano feature con importanza relativa bassa, beneficiano della loro combinazione con altre variabili.

Inoltre, la distribuzione delle feature nel dataset presenta strutture di correlazione che la semplice selezione basata su Random Forest non ha considerato. Di conseguenza, il modello risultante è meno robusto, mostrando una maggiore confusione tra classi simili. Dopo questi test, si è deciso di annullare il tentativo di riduzione delle feature e mantenere il modello Random Forest con tutte le feature disponibili, garantendo così prestazioni più stabili e una classificazione più accurata.

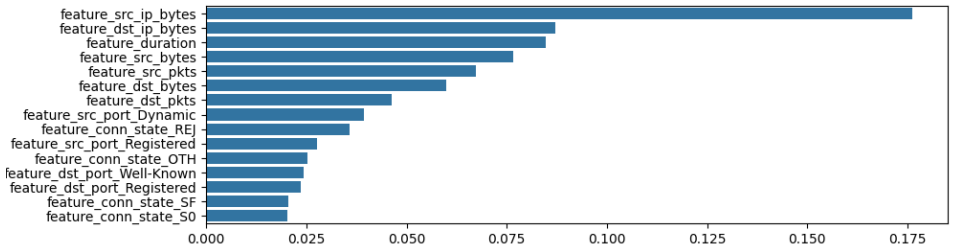


Fig. 2.18: Importanza delle feature

2.1.3 KNN

Implementazione e Addestramento del K-Nearest Neighbors (KNN)

Il modello *K-Nearest Neighbors* (KNN) è stato implementato utilizzando la libreria `scikit-learn`, valutando diverse strategie di preprocessamento e configurazioni iperparametriche per ottimizzare le prestazioni. Data la sensibilità del KNN alla scala delle feature e alla presenza di outlier, sono state applicate differenti tecniche di normalizzazione, gestione degli outlier e riduzione della dimensionalità.

Preparazione del dataset Anche per il modello K-Nearest Neighbors (KNN) segue i passaggi descritti nei paragrafi 1.3 e 1.4.
Si è stata effettuata una pulizia rigorosa del dataset, eliminando le feature ridondanti o poco informative, inoltre si è eseguita la disambiguazione dei valori della colonna `service`. Le variabili categoriche con elevata cardinalità sono state rimosse per evitare di introdurre una dimensionalità eccessiva, mentre le restanti sono state codificate riducendo il numero di valori, infine le variabili booleane sono state trasformate in valori binari per garantire un'interpretazione univoca.
Il modello è stato testato sia con che senza diverse tecniche di sostituzione dei valori nulli, sono state utilizzate due differenti tecniche, una che prevedeva la sostituzione solo dei valori –, mentre l'altra che prevedeva la sostituzione sia dei valori - che dei valori \,
Poiché KNN è un algoritmo basato sulla distanza, sono state applicate tecniche di normalizzazione delle feature (*l1, l2*), queste sono risultate essenziali per evitare che variabili con scale diverse potessero dominare il calcolo della distanza. Sono stati testati anche diversi metodi di scaling, tra cui *Standard Scaling*, *MinMax Scalin*g e *Quantile Scaling*, al fine di determinare l'approccio più efficace. Inoltre, per ogni scaler sono stati utilizzati diversi outlier per testare l'impatto che questi avrebbero avuto sui diversi metodi.
Per ridurre la dimensionalità e migliorare l'efficienza computazionale, è stata applicata la PCA con una soglia del 99%, LDA ed il modello è stato anche testato in assenza di tecniche di riduzione della dimensionalità.

Addestramento e tuning degli iperparametri

L'addestramento è stato condotto su cinque dataset bilanciati, derivanti dal processo di preprocessing, con un numero fisso di campioni per garantire uniformità nell'analisi comparativa. La selezione dei parametri ottimali è stata effettuata tramite grid search, considerando i seguenti iperparametri:

- **Numero di vicini** (`n_neighbors`): $\{3, 5, 10, 15, 20\}$.
- **Metrica di distanza**: *Euclidean*, *Minkowski*.

- **Schema di pesatura**: *uniform* (peso uguale per tutti i vicini) e *distance* (peso inversamente proporzionale alla distanza).

Tipo di Iperparametro	Valori Testati
dataset	dataset_1, dataset_2, dataset_3, dataset_4, dataset_5
target count	15000
scaling methods	quantile, standard, minmax, l1, l2
outlier methods	no, base, isolation_forest, percentile, dynamic_threshold
dimensionality reduction	PCA, LDA, None
PCA threshold	0.85, 0.93, 0.95, 0.99
n_neighbors	2, 3, 5, 10, 15, 20
metric	euclidean, minkowski
weights	uniform, distance
replace_value	no, mode, mode_all

Tab. 2.3: Iperparametri testati

Risultati del tuning L'analisi delle prestazioni del modello K-Nearest Neighbors ha evidenziato il ruolo cruciale della scelta degli iperparametri e delle tecniche di preprocessing nel miglioramento della classificazione.

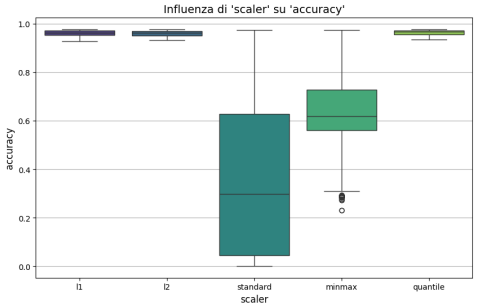


Fig. 2.19: Performance rispetto agli scaler

Uno dei primi aspetti esaminati è stato l'effetto della normalizzazione e dello scaling (2.19). Le statistiche mostrano chiaramente che gli scaler *MinMax* e *Standard* hanno prodotto risultati significativamente inferiori rispetto agli altri metodi testati, con mediane rispettivamente di 0.620 e 0.298, e un primo quartile molto basso (0.560 e 0.045). Per questo motivo, questi scaler sono stati esclusi dalle successive analisi. Al contrario, le normalizzazioni *L1*, *L2* e *Quantile* hanno fornito prestazioni nettamente migliori, con mediane pari a 0.963, 0.962 e 0.969 rispettivamente, e una minore variabilità nei risultati. Tra queste, *L1* ha ottenuto i valori più elevati, risultando la normalizzazione più efficace.

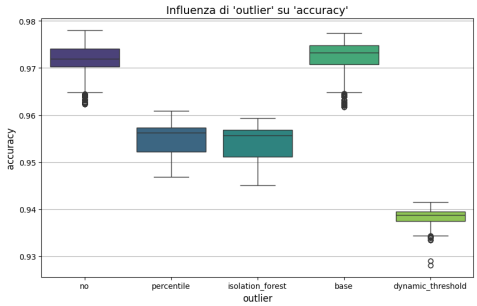


Fig. 2.20: Performance rispetto agli outlier

Successivamente, è stata valutata l'influenza della rimozione degli outlier (2.20). Dai risultati emerge che l'eliminazione dei soli valori estremi, come effettuato con *Dynamic Threshold*, *Percentile* e *Random Forest*, ha comportato una riduzione significativa delle prestazioni. Ad esempio, *Dynamic Threshold* ha ottenuto una mediana di 0.938, nettamente inferiore rispetto a *No outlier removal* (0.972) e alla rimozione base (0.973). Pertanto, questi metodi di rimozione degli outlier sono stati scartati, mentre le migliori prestazioni sono state ottenute senza alcuna rimozione o con una rimozione mirata dei dati più anomali.

Per quanto riguarda la riduzione della dimensionalità, i risultati dimostrano che i modelli addestrati con *LDA* superano quelli basati su *PCA* (2.21). Le statistiche confermano questa tendenza: *LDA* ha ottenuto una mediana di 0.955, superiore a *PCA* (0.944), suggerendo una maggiore capacità di conservare le informazioni discriminanti tra le classi. Inoltre, test preliminari sulla soglia di varianza di *PCA* hanno permesso di fissarla a 0.99, poiché soglie inferiori comportavano un drastico calo delle prestazioni.

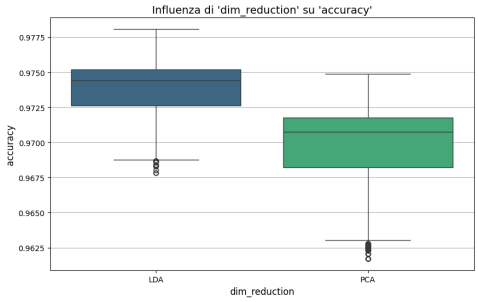


Fig. 2.21: Performance rispetto alla riduzione della dimensionalità

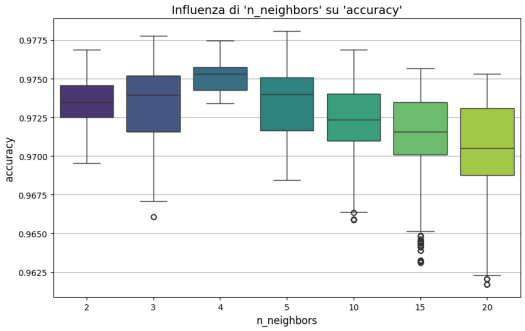


Fig. 2.22: Performance rispetto a n_neighbors

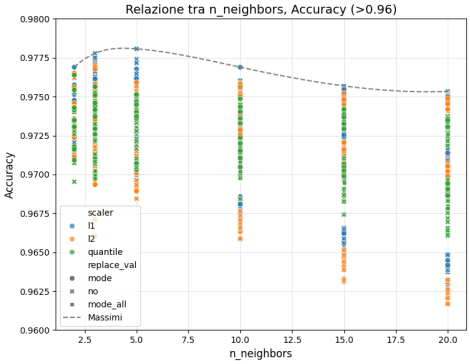


Fig. 2.23: Performance rispetto ai vicini con valutazione rispetto agli scaler e alla sostituzione dei valori di default

Un'analisi dettagliata dell'iperparametro `n_neighbours` ha mostrato che l'accuratezza segue una distribuzione a campana (2.22): le migliori prestazioni si ottengono con `n = 5`, mentre valori più alti o più bassi determinano un progressivo decadimento delle performance. Questo comportamento è coerente con la natura del KNN, in cui un numero eccessivo di vicini introduce rumore, mentre un numero troppo basso riduce la capacità di generalizzazione.

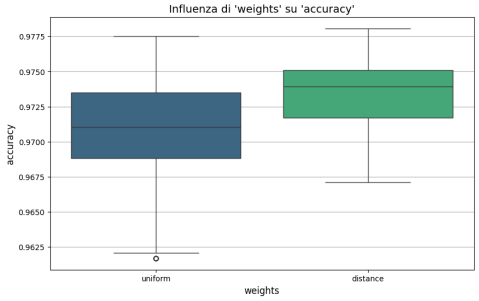


Fig. 2.24: Performance rispetto a weights

Un altro parametro di rilievo è stato il metodo di ponderazione dei vicini. I modelli addestrati con `weights=distance` hanno ottenuto prestazioni leggermente migliori rispetto a `weights=uniform`, con una mediana di 0.974 rispetto a 0.971 (2.24). Questo suggerisce che attribuire un peso maggiore ai vicini più prossimi migliora la capacità predittiva del modello. Le osservazioni sugli iperparametri `weights` e `n_neighbors` sono evidenti anche nella heatmap in figura 2.25 che mostra il valore massimo di accuratezza per valori di c e weights.

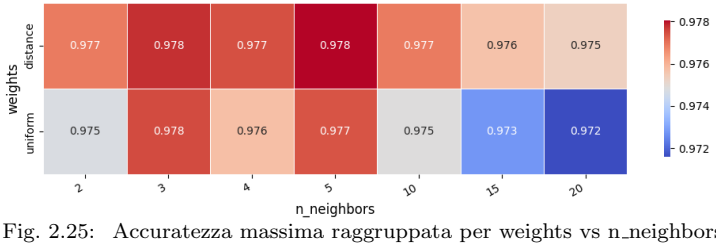


Fig. 2.25: Accuratezza massima raggruppata per weights vs n_neighbors

L'iperparametro `metric` (2.27), che distingue tra le metriche di distanza *Euclidea* e *Minkowski*, non ha mostrato variazioni significative nelle prestazioni, indicando che entrambe le metriche risultano equivalenti per questo problema di classificazione.

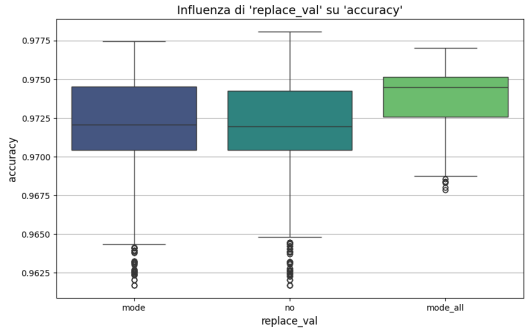


Fig. 2.26: Performance rispetto alla gestione dei valori mancanti

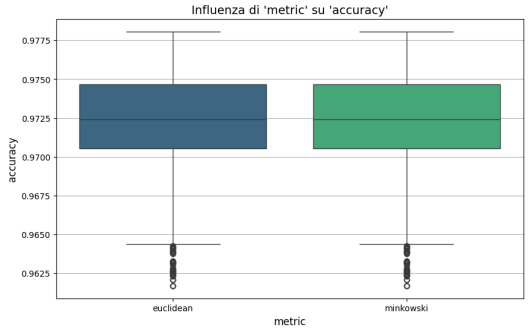


Fig. 2.27: Performance rispetto a metric

Infine, l'analisi ha anche considerato l'impatto della gestione dei valori mancanti (2.26). I migliori risultati sono stati ottenuti senza alcun rimpiazzo dei valori nulli, seguiti dalla sostituzione con la media del protocollo di appartenenza. Al contrario, la sostituzione con la media dell'intera colonna ha portato a un leggero peggioramento delle prestazioni, indicando che questa strategia potrebbe introdurre rumore nei dati piuttosto che migliorare la generalizzazione del modello.
L'analisi del tuning ha permesso di identificare una configurazione ottimale per il modello KNN. Le migliori prestazioni sono state ottenute con la normalizzazione *L1*, senza rimozione degli outlier, con `n_neighbors` pari a 5, utilizzando *LDA* come tecnica di riduzione della dimensionalità e `weights=distance`. Inoltre, i risultati suggeriscono che la gestione dei valori nulli gioca un ruolo chiave, con la mancata sostituzione dei valori nulli che ha fornito risultati leggermente migliori. L'iperparametro `metric` non ha influenzato significativamente le prestazioni del modello.

2.2 Machine Learning Supervised su reti neurali ed architettura Feed-Forward

2.2.1 Reti

L'obiettivo primario di questa fase è stato progettare e valutare un'architettura di rete neurale in grado di eseguire una classificazione multiclasse degli attacchi con elevata precisione e stabilità. Per tale scopo, è stata sviluppata un'architettura, denominata FeedForwardPlus, implementata in PyTorch per garantire flessibilità e modularità.

Preparazione del dataset La preparazione del dataset per le reti neurali ha seguito un approccio differente rispetto ai modelli tradizionali, rimuovendo un numero minore di colonne: limitandosi esclusivamente alle variabili non adatte al task di addestramento, come `ts` o `label`, oltre alle variabili categoriche con elevata cardinalità di valori unici, che avrebbero comportato un aumento eccessivo della dimensionalità in fase di encoding.
Analogamente agli altri preprocessing la colonna `service` è stata trasformata in una

rappresentazione binaria rispetto ad ogni servizio, mentre gli indirizzi IP sono stati categorizzati in classi e le porte di comunicazione suddivise nei tre intervalli standard (Well-Known, Registered e Dynamic). Infine le variabili booleane sono state uniformate ai valori True/False.
Per la gestione dei valori di default, analogamente agli altri modelli, si è utilizzata sia la sostituzione di - con la moda che la sostituzione di - e \. In questo caso si è preferito concentrarsi sulla normalizzazione, testando con metodi di scaling usando solo minmax, sono stati invece utilizzate le diverse tecniche d'outlier per comprendere il loro impatto sui differenti scaler.
Per un'analisi approfondita si è scelto di testare entrambe le tecniche di riduzione della dimensionalità oltre che l'assenza di queste, poiché le reti neurali sono in grado di apprendere rappresentazioni latenti direttamente dai dati grezzi, sfruttando la profondità della rete per estrarre caratteristiche significative.

Definizione dell'Architettura La rete FeedForwardPlus è stata sviluppata con l'obiettivo di garantire flessibilità nella configurazione dei suoi parametri principali, consentendo di adattare l'architettura alle specifiche esigenze del problema di classificazione. Il modello è composto da uno strato di input, che accetta il numero di feature del dataset, una serie di strati nascosti configurabili e uno strato di output che restituisce la predizione finale. La dimensione dei livelli nascosti, definita dal parametro hidden_size, determina il numero di neuroni presenti in ciascun livello intermedio, influenzando la capacità rappresentativa del modello. La profondità della rete, regolata dal parametro depth, consente di variare il numero di strati nascosti, permettendo di esplorare diverse configurazioni architetturali.

Per migliorare la stabilità e la velocità della convergenza, è possibile introdurre la normalizzazione batch, attivabile attraverso il parametro batch_norm, che standardizza le attivazioni a ogni livello e riduce la dipendenza dalla distribuzione dei dati di input. Inoltre, per mitigare il rischio di overfitting, è stato integrato un meccanismo di dropout, configurabile tramite il parametro drop, che introduce una probabilità di disattivazione casuale di alcuni neuroni durante l'addestramento.

L'architettura della rete è implementata in modo modulare attraverso la classe nn.Sequential, che consente di costruire dinamicamente la sequenza di strati in base ai parametri selezionati. In particolare, vengono gestite tre configurazioni possibili: una rete senza normalizzazione né dropout, una con normalizzazione batch e una con dropout attivo. Questa modularità permette di confrontare facilmente gli effetti di ciascun meccanismo sulle prestazioni del modello, ottimizzando la scelta della configurazione migliore per il task di classificazione.

Tuning degli iperparametri

Il processo di tuning degli iperparametri è stato condotto per ottimizzare le prestazioni del modello, combinando tecniche di addestramento mirate e un'esplorazione sistematica delle configurazioni. L'addestramento è stato eseguito utilizzando la funzione di perdita *Cross Entropy Loss*, particolarmente adatta per problemi di classificazione multiclasse. Gli algoritmi di ottimizzazione `Adam` e `SGD` sono stati confrontati empiricamente, e `Adam` è stato selezionato date le performance nettamente superiori. Durante il training, è stato

utilizzato uno scheduler del tasso di apprendimento (`StepLR`) per ridurre progressivamente il learning rate, garantendo un apprendimento più accurato nelle fasi avanzate. In aggiunta, il modello è stato valutato su diverse tecniche di riduzione della dimensionalità e metodi di scaling per garantire una corretta rappresentazione dei dati e migliorare la stabilità durante l'addestramento.

La fase di tuning è stata implementata attraverso una grid search su un set predefinito di iperparametri, mirata a esplorare combinazioni che massimizzassero le prestazioni del modello. Ogni combinazione di iperparametri è stata valutata attraverso un processo iterativo che prevedeva l'addestramento della rete e la sua validazione su un dataset indipendente. Durante ciascun esperimento, sono state monitorate metriche come accuratezza, F1-score, precision e recall per identificare la configurazione ottimale.

Tipo di Iperparametro	**Valori Testati**
dataset	dataset_1, dataset_2, dataset_3, dataset_4, dataset_5
target count	15000, 20000, 25000
scaling methods	minmax, l1, l2
outlier methods	no, percentile, isolation_forest, base, dynamic_threshold
dimensionality reduction	PCA, LDA, None
hidden_size	16, 32, 64, 128, 192, 256, 512
dropout	0.0, 0.1, 0.2, 0.3, 0.4, 0.5
depth	2, 3, 4, 5, 6, 7, 8, 9, 10
gamma	0.1, 0.3, 0.4, 0.5, 0.8, 0.85, 0.9, 1.0
learning_rate	1.0e-05, 1.0e-04, 1.0e-03, 1.0e-02, 2.0e-03, 2.0e-05, 5.0e-04, 5.0e-05, 5.0e-03, 9.0e-04, 1.2e-03, 1.0e-01
batch_size	16, 32, 64, 128
batch_norm	True, False
step_size	7.5, 10, 12, 12.5, 15, 20, 22, 25, 50
weight_decay	1e-06, 1e-05, 0.001, 0.1
replace_value	no, mode, mode_all

2.2.2 Risultati

L'analisi condotta ha avuto l'obiettivo di ottimizzare le prestazioni delle reti neurali attraverso la selezione dei migliori iperparametri e tecniche di preprocessing.

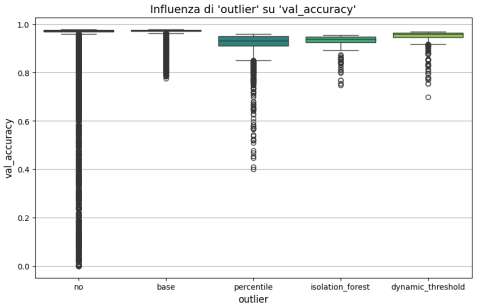


Fig. 2.28: Performance rispetto agli outliers

In primo luogo, è stata analizzata l'influenza della gestione degli outlier sulle prestazioni del modello. L'eliminazione aggressiva degli outlier mediante tecniche quali *dynamic threshold*, *percentile* e *random forest* (2.28) ha determinato un peggioramento delle performance della rete, suggerendo che la perdita di dati informativi penalizzi l'apprendimento. Al contrario, le configurazioni che hanno mantenuto l'intero dataset o che hanno escluso solo gli outlier più estremi hanno fornito risultati superiori.

Per quanto riguarda la normalizzazione e lo scaling, le trasformazioni basate sulle norme L_1 e L_2 si sono rivelate le più efficaci, mentre approcci come lo *scaling min-max* hanno mostrato un impatto negativo sulle prestazioni complessive del modello.

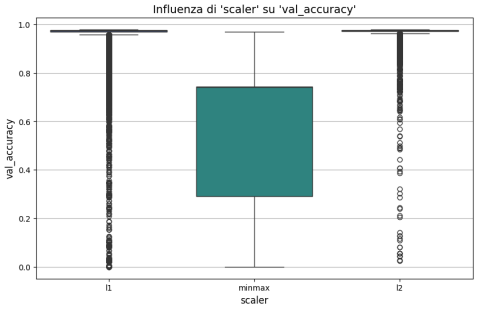


Fig. 2.29: Performance rispetto allo scaler

Un aspetto critico è stato l'effetto della riduzione della dimensionalità. L'analisi ha mostrato che l'uso della *Linear Discriminant Analysis* (LDA) migliora l'addestramento del modello, mentre la *Principal Component Analysis* (PCA) ha mostrato risultati inferiori, con una soglia ottimale di varianza fissata a 0.97. L'assenza di qualsiasi riduzione della dimensionalità ha prodotto risltati leggermente inferiori rispetto a quelli calcolati con LDA, confermando il vantaggio di tale tecnica nell'estrazione delle caratteristiche più discriminanti (2.30).

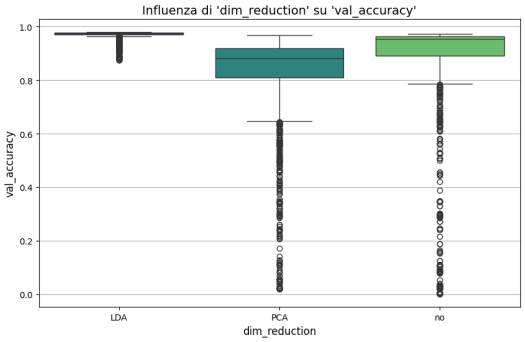


Fig. 2.30: Performance rispetto alla riduzione dimensionale

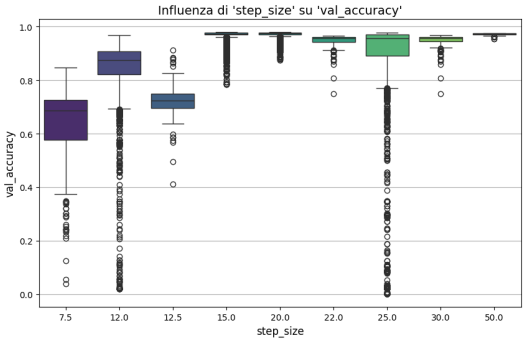


Fig. 2.31: Performance rispetto al step size

Il valore ottimale del *learning rate* è stato identificato in 0.001, con una riduzione delle prestazioni sia per valori inferiori che superiori a questa soglia (2.41). Il dropout, utilizzato per la regolarizzazione, ha mostrato un comportamento ottimale con valori pari a 0.1, mentre *dropout* più elevati hanno causato un eccessivo indebolimento della rete (2.40).

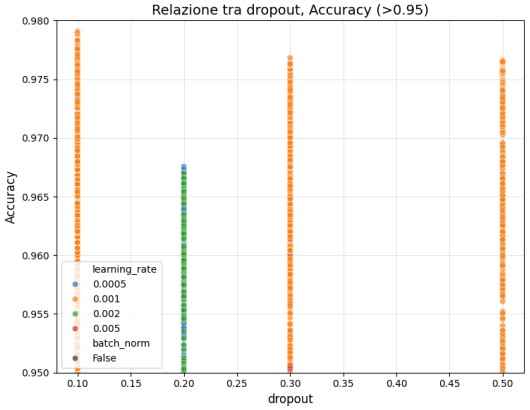


Fig. 2.32: Performance rispetto al dropout e learning rate rispetto all'accuracy

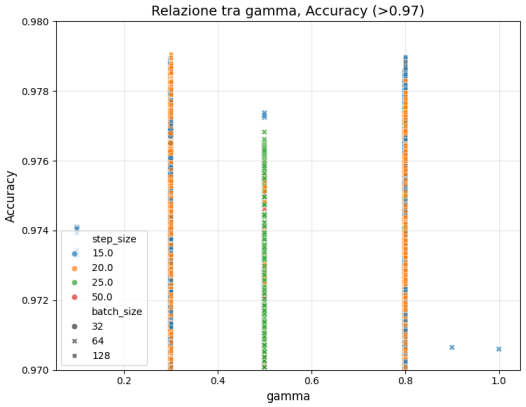


Fig. 2.33: Performance al gamma e al batch size rispetto all'accuracy

Il parametro γ ha influenzato le prestazioni del modello, con valori ottimali individuati in 0.3 e 0.8, che hanno garantito una distribuzione più concentrata e stabile delle accuratezze, metre l'analisi della dimensione dello *step size* ha evidenziato che i valori

ottimali sono 15 e 20, mentre scelte al di fuori di questo intervallo hanno determinato una riduzione della qualità del modello (2.33).

Per quanto riguarda la dimensione dello strato nascosto, è emerso che valori inferiori a 64 risultano subottimali, mentre le configurazioni con *hidden size* pari a 64, 128, 256 e 512 hanno mostrato le migliori prestazioni (2.34). La profondità della rete si è rivelata un parametro critico, con le migliori performance ottenute per reti di profondità pari a 2, 3 o 5 strati, mentre un numero maggiore di strati ha introdotto instabilità nell'addestramento e un degrado complessivo delle prestazioni.

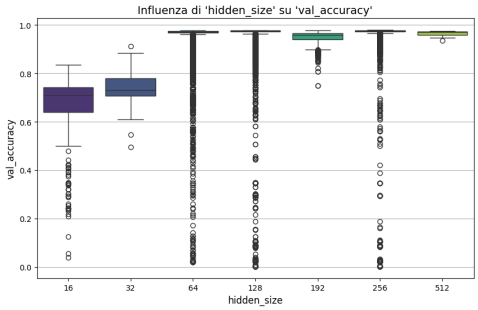


Fig. 2.34: Performance rispetto all'hidden size

L'analisi della dimensione del *batch* ha dimostrato che i valori ottimali sono 32, 128 e 256, mentre il valore 16 ha mostrato una riduzione delle performance (2.37). Un altro elemento significativo è stato il ruolo della *batch normalization*, il cui utilizzo ha portato a un impatto negativo sulle prestazioni (2.36).

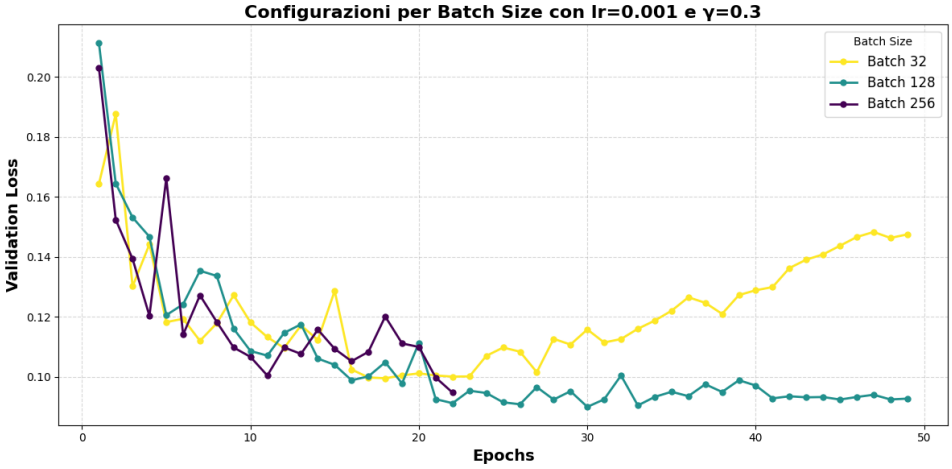


Fig. 2.35: Andamento dell'accuracy rispetto al batch al variare dell'epoca mantendo fisso lr e gamma

Il grafico rappresenta l'andamento della validation loss per diversi valori del batch size, mantenendo invariati gli altri iperparametri. Si osserva che un batch size più piccolo (32) porta a una rapida discesa iniziale della loss, ma successivamente mostra un chiaro segnale di overfitting, con una risalita progressiva dopo circa 20 epoche. Questo comportamento è dovuto alla maggiore variabilità degli aggiornamenti dei pesi, che permette una rapida esplorazione dello spazio dei parametri ma rende il modello più sensibile al rumore.
Il batch size 128 si distingue per una convergenza più stabile e una validation loss complessivamente più bassa, suggerendo un buon bilanciamento tra velocità di apprendimento e capacità di generalizzazione. Infatti, con batch medi, la stima del gradiente è meno rumorosa rispetto ai batch piccoli, ma mantiene una sufficiente diversità negli aggiornamenti per evitare soluzioni subottimali.
Un batch size di 256 potrebbe risultare troppo grande per questo specifico dataset, contribuendo all'instabilità osservata. Quando il batch è molto ampio, il numero di aggiornamenti dei pesi per epoca si riduce drasticamente, rallentando l'adattamento del modello

e rendendolo meno reattivo alle variazioni nei dati. Inoltre, nei dataset sbilanciati, un batch grande potrebbe amplificare il problema della predominanza delle classi maggioritarie, portando a una stima del gradiente meno rappresentativa della distribuzione complessiva.

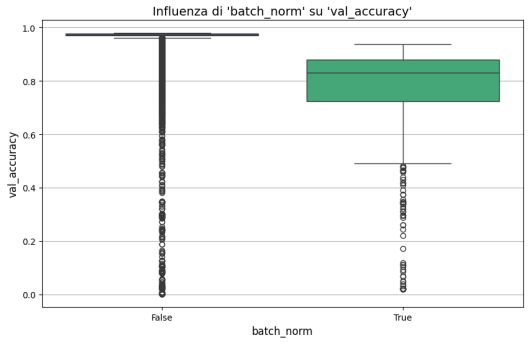


Fig. 2.36: Performance rispetto al batch normalization

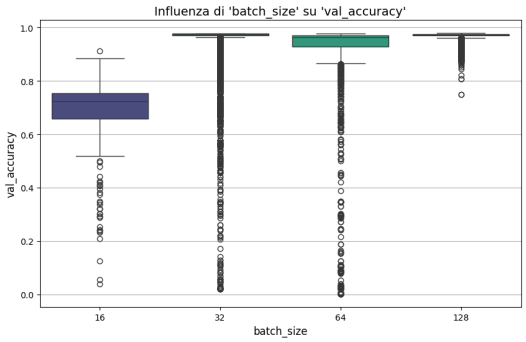


Fig. 2.37: Performance rispetto al batch size

Dai risultati ottenuti emerge che la configurazione ottimale prevede l'uso di *LDA* per la riduzione della dimensionalità, la normalizzazione con norme L_1 o L_2, un *learning rate* pari a 0.001, un *dropout* di 0.1, *gamma* pari a 0.3 o 0.8, *hidden size* tra 64 e 512, una profondità della rete di 2 o 3 strati, e una *batch size* di 32, 128 o 256.

Restringendo l'analisi a queste configurazioni ottimali, è stato studiato l'impatto della gestione dei valori nulli. Si è osservato che la sostituzione dei valori mancanti con la moda calcolata sulle sole colonne del protocollo produce risultati migliori rispetto alla sostituzione basata su tutte le colonne del dataset. Infine, l'analisi dell'effetto del *weight decay* ha evidenziato un impatto negativo sulle prestazioni, con una riduzione dell'accuratezza rispetto alla configurazione in cui tale parametro è impostato a zero.

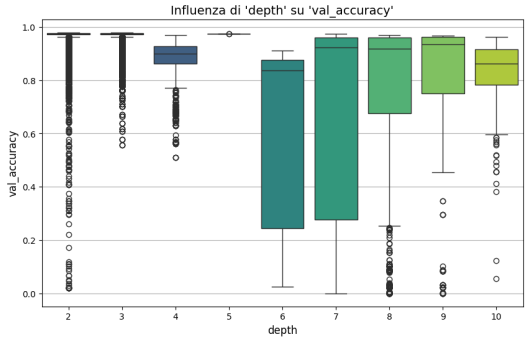


Fig. 2.38: Boxplot delle performance rispetto alla profondità della rete

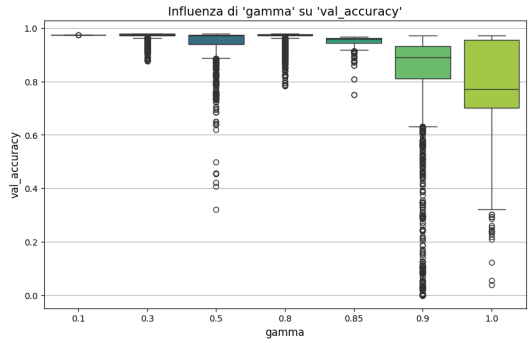


Fig. 2.39: Boxplot delle performance rispetto al gamma

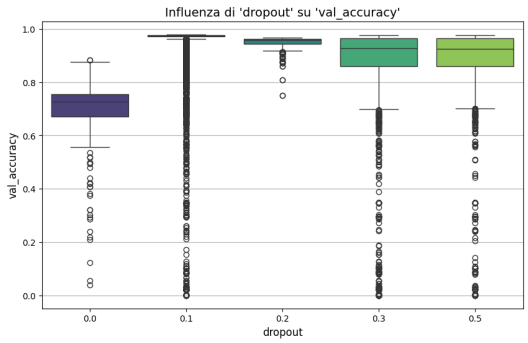


Fig. 2.40: Boxplot delle performance rispetto al dropout

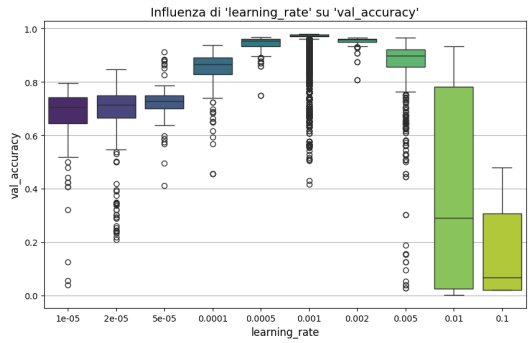


Fig. 2.41: Boxplot delle performance rispetto al learning rate

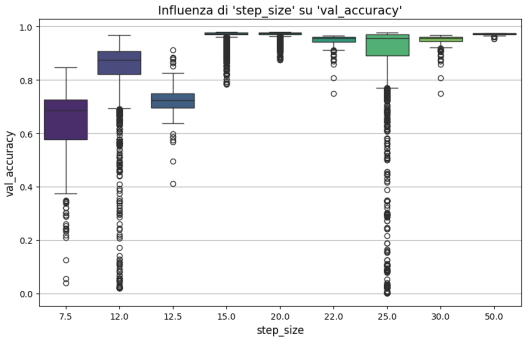


Fig. 2.42: Boxplot delle performance rispetto al step size

2.2.3 Architettura alternativa

Nell'ambito della riduzione della dimensionalità, abbiamo esplorato un'architettura alternativa basata su reti neurali, invece di utilizzare tecniche come PCA o LDA. L'obiettivo era verificare se una riduzione progressiva del numero di neuroni nei livelli nascosti potesse mantenere una buona capacità di rappresentazione dei dati e migliorare le prestazioni. Abbiamo sperimentato diverse configurazioni di rete, variando i seguenti parametri:

Parametro	Valore
Numero di target	20.000
Scaler	L1
Gestione outlier	Base
Gamma	0.3, 0.8, 1.0
Learning rate	0.0005, 0.001, 0.01
Batch size	32, 64, 128
Step size	15, 20

Tab. 2.4: Parametri di configurazione

Struttura della rete (hidden_size)
[128, 64, 32]
[64, 32, 16]
[128, 64, 32, 16]
[256, 64, 16]
[256, 128, 64, 32, 16]
[128, 32, 8]

Tab. 2.5: Configurazioni della struttura della rete

I risultati ottenuti per ogni configurazione di hidden_size sono riportati nella tabella seguente:

Hidden Size	Min	Q1	Mediana	Q3	Max
[128, 32, 8]	0.6606	0.9069	0.9429	0.9579	0.9680
[128, 64, 32, 16]	0.8220	0.9314	0.9480	0.9579	0.9677
[128, 64, 32]	0.7924	0.9561	0.9651	0.9687	0.9732
[256, 128, 64, 32, 16]	0.0836	0.9386	0.9520	0.9652	0.9681
[256, 64, 16]	0.8039	0.9465	0.9584	0.9657	0.9721
[64, 32, 16]	0.7392	0.9185	0.9329	0.9415	0.9460

Tab. 2.6: Prestazioni delle diverse arch

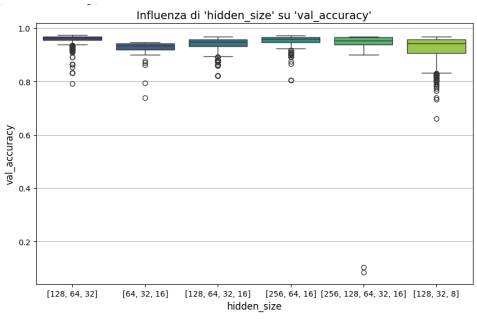


Fig. 2.43: Performance rispetto a hidden_size

Dall'analisi dei risultati emerge che l'architettura con hidden size [128, 64, 32] ha ottenuto le migliori prestazioni complessive, con una mediana di 0.9651 e un valore massimo di 0.9732. Tuttavia, altre configurazioni come [256, 64, 16] e [128, 64, 32, 16] hanno raggiunto performance comparabili.

Si nota che la configurazione [256, 128, 64, 32, 16] ha un valore minimo estremamente basso

(0.0836), suggerendo instabilità nella rete per alcuni scenari. Allo stesso tempo, la configurazione più semplice [64, 32, 16] ha mostrato prestazioni inferiori rispetto alle alternative più complesse.

Nonostante alcune reti abbiano ottenuto buoni risultati, questa metodologia non ha raggiunto livelli di performance comparabili a quelli ottenuti con altre tecniche di riduzione della dimensionalità, come PCA o LDA. Ciò suggerisce che la riduzione dimensionale basata su reti neurali potrebbe non essere la strategia ottimale per questo dataset specifico o che potrebbero essere necessarie ulteriori ottimizzazioni.

2.3 Machine Learning Supervised con modelli deep per Tabular Data

2.3.1 TabTransform

Implementazione di TabTransform L'implementazione del modello TabTransformer è stata realizzata utilizzando PyTorch, con l'obiettivo di sfruttare le capacità delle reti Transformer per l'elaborazione di dati tabulari. È stata creata la classe che implementa l'interfaccia Dataset di PyTorch che converte i dati in tensori e li rende compatibili con il framework di deep learning. L'architettura del TabTransformer è composta da tre componenti principali:

- **Layer di embedding**: un livello lineare che proietta le feature di input in uno spazio a dimensione ridotta (dim_embedding)
- **Trasformazione tramite Transformer**: Un encoder Transformer, basato su più livelli (num_layers) con self-attention multi-head (num_heads)
- **Classificatore finale**: un livello lineare che mappa l'output del Transformer allo spazio delle classi di output.

Preparazione del dataset La preparazione del dataset per il modello TabTransformer ha seguito un approccio simile a quello adottato per altri modelli: anche in questo caso, si è proceduto con una pulizia delle colonne meno stringente, eliminando esclusivamente le variabili che avrebbero potuto compromettere le performance del modello, mantenendo così più alto il numero di variabili informative. La colonna `service` è stata trasformata in una rappresentazione binaria, mentre gli indirizzi IP sono stati categorizzati in classi. Le porte di comunicazione sono state suddivise nei tre intervalli standard (Well-Known, Registered e Dynamic), e le variabili booleane sono state uniformate ai valori True/False.
Analogamente a quanto fatto per gli altri modelli, la gestione dei valori mancanti è stata effettuata sostituendo i valori mancanti (indicati come –) con la moda della colonna corrispondente. Per quanto riguarda la normalizzazione, si è scelto di utilizzare esclusivamente tecniche di normalizzazioni e di testare diverse tecniche per la rimozione degli outlier, uesto approccio ha permesso di analizzare un numero maggiore di iperparametri e di valutare quale combinazione di tecniche produca i migliori risultati in termini di performance. Inoltre, sono stati analizzati i possibili impatti della riduzione della dimensionalità sul modello, testando sia l'efficacia di Principal Component Analysis (PCA) che di Linear Discriminant Analysis (LDA) come tecniche di riduzione dimensionale, al

fine di determinare quale approccio consentisse di migliorare le prestazioni del modello riducendo la complessità computazionale.

Addestramento e tuning degli iperparametri L'addestramento è stato condotto su cinque dataset bilanciati, derivanti dal processo di preprocessing, con un numero fisso di campioni che è stato variato per analizzare prima la dimensione di 15000 e poi 20000. Inizialmente è stata effettuata una valutazione usando 50 epoche effettuata tramite grid search, considerando i seguenti iperparametri:

- **Batch size**: definisce il numero di campioni da elaborare in ogni batch durante l'addestramento.
- **Learning rate**: specifica la velocità con cui i pesi del modello vengono aggiornati durante l'ottimizzazione. Valori più bassi rallentano l'apprendimento, mentre valori più alti accelerano l'apprendimento.
- **Number of epochs**: definisce il numero di volte in cui il modello passa attraverso l'intero dataset durante l'addestramento.
- **Patience**: specifica il numero di epoche consecutive senza miglioramenti nella perdita di validazione prima di fermare l'addestramento (early stopping).
- **Dimensionality of embedding**: Imposta la dimensione degli spazi di embedding, ovvero la rappresentazione densa delle variabili categoriche.
- **Number of heads**: definisce il numero di "heads" nel meccanismo di attenzione del modello Transformer.
- **Number of layers**: determina il numero di strati (layers) del modello.
- **Gamma**: parametro dello scheduler StepLR, testato per regolare la velocità di decadimento del learning rate, testato poi tenendo in considerazione anche i risultati ottenuti nelle reti.
- **Step size**: numero di epoche dopo il quale lo scheduler modifica il tasso di apprendimento, testato poi tenendo in considerazione anche i risultati ottenuti nelle reti.

Tipo di Iperparametro	Valori Testati
Scaler	l1, l2
Target count	15000, 20000
Outlier	no, base, dynamic_threshold, isolation_forest, percentile
Dimensionality reduction	LDA, PCA
PCA threshold	NaN, 0.995, 0.999
Batch size	16, 32, 64, 128, 256, 512
Learning rate	0.0009, 0.00095, 0.001, 0.0011, 0.01
Number of epochs	20, 30, 50, 80, 100, 150
Patience	10, 20
Dimensionality of embedding	16, 32, 64, 128
Number of heads	2, 4, 8
Number of layers	2, 3, 4, 5
Gamma	0.3, 0.9
Step size	10, 15, 18, 20, 22
Replace value	no, mode

Tab. 2.7: Iperparametri testati

Risultati

Nella fase di data cleaning, sono state testate diverse strategie per la gestione degli outlier, confrontando l'effetto della loro rimozione rispetto al mantenimento nei dati di addestramento. Dai risultati emerge che l'approccio migliore consiste nel mantenere gli outlier oppure rimuoverli solo se superano soglie molto ampie.

L'analisi delle performance ha evidenziato che la rimozione degli outlier con metodi come Isolation Forest e Percentile fornisce risultati leggermente inferiori rispetto al mantenimento dei dati originali (2.45). Per verificare l'effetto del bilanciamento delle classi, sono stati testati due scenari: uno con 15.000 istanze per classe e un altro con 20.000. I risultati mostrano che il modello riesce a generalizzare meglio con 20.000 istanze per classe, garantendo una maggiore stabilità delle prestazioni e una minore varianza tra le diverse metriche valutate.

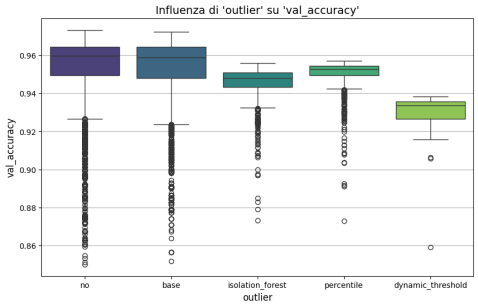


Fig. 2.44: Performance rispetto agli outlier

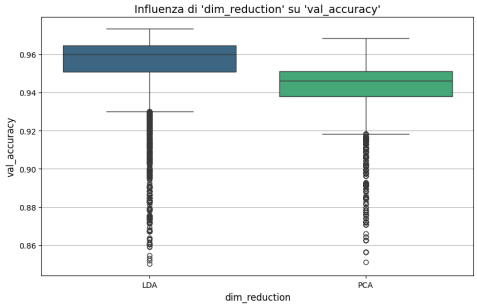


Fig. 2.45: Performance rispetto alla riduzione di dimensionalità

La riduzione della dimensionalità ha portato a un miglioramento nelle prestazioni del modello. In particolare, LDA (Linear Discriminant Analysis) ha mostrato risultati migliori rispetto a PCA (Principal Component Analysis), suggerendo che l'analisi delle componenti lineari aiuta a catturare meglio le informazioni rilevanti per la classificazione (2.45). Il confronto tra normalizzazione L1 e L2 ha mostrato che la normalizzazione L1 produce performance superiori. Questo suggerisce che la somma dei valori assoluti utilizzata da L1 preserva meglio le caratteristiche del dataset rispetto alla normalizzazione L2, che enfatizza maggiormente le deviazioni minime nei dati.

L'analisi delle prestazioni del modello rispetto alla struttura della rete ha mostrato che l'uso di 2 o 3 layer fornisce risultati ottimali (2.46). Il numero di teste non sembra avere un impatto significativo, anche se 2 e 8 teste permettono performance leggermente migliori (2.47). Inoltre, la dimensione ottimale dell'embedding è risultata essere 64 (2.48)., poiché garantisce un buon equilibrio tra capacità di rappresentazione e riduzione della complessità computazionale.

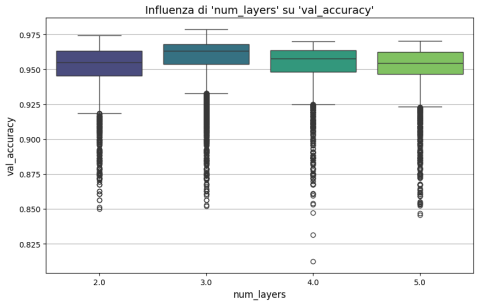


Fig. 2.46: Performance rispetto al numero di livelli

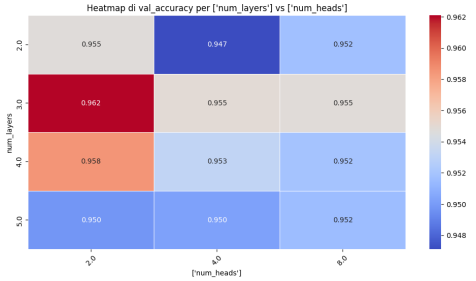


Fig. 2.47: Accuracy media raggruppata per numero di layer vs numero di teste di attenzione

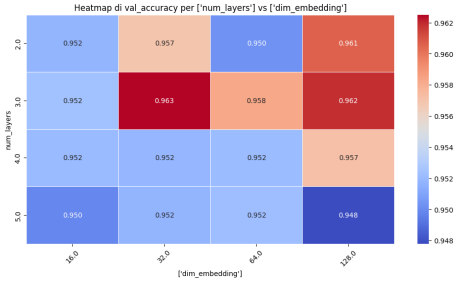


Fig. 2.48: Accuracy media raggruppata per numero di layer rispetto alla dimensione dell'embedding

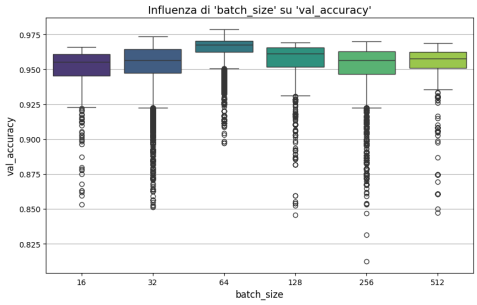


Fig. 2.49: Performance rispetto al batch

L'uso della tecnica di mini-batch è stato valutato con diverse dimensioni per trovare il compromesso ottimale tra efficienza computazionale e performance. La dimensione ottimale del mini batch è risultata essere 64 (2.49), in quanto permette di ottenere prestazioni più elevate rispetto a batch più piccoli o più grandi, garantendo una convergenza più stabile durante l'addestramento.

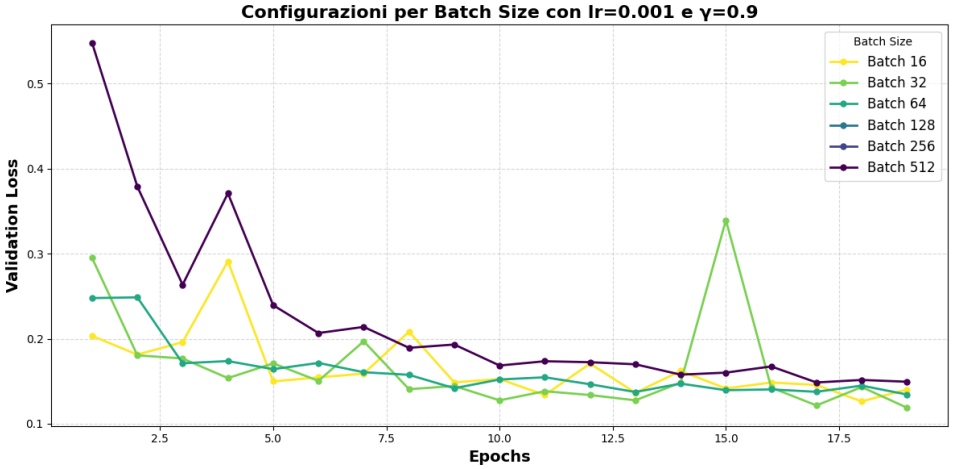


Fig. 2.50: Andamento dell'accuracy rispetto al batch al variare dell'epoca mantendo fisso lr e gamma

Come nel caso delle reti, il batch presenta un comportamento controintuitivo, infatti il batch size 64 mostra la convergenza più stabile (2.50), con una loss significativamente più bassa rispetto agli altri batch. Questo suggerisce che, per questa specifica configurazione e dataset, un batch relativamente piccolo permette un aggiornamento più efficace dei pesi. Per batch più grandi (128, 256 e 512), la convergenza è meno regolare, con oscillazioni più evidenti nelle prime epoche. In particolare, il batch 512 mostra un picco iniziale molto alto e una discesa più caotica, indicando una difficoltà iniziale nel trovare una traiettoria di apprendimento stabile. Questo potrebbe essere dovuto al fatto che un batch così grande riduce la frequenza degli aggiornamenti dei pesi, rallentando l'adattamento del modello nelle prime fasi.

Tra i tassi di apprendimento testati, il valore di 0.001 è quello che fornisce le migliori prestazioni (2.51). Inoltre, è stato introdotto un fattore gamma che degrada il learning rate ogni determinato numero di epoche. Il valore ottimale di gamma è stato individuato in 0.3 (2.52) con un degrado ogni 15-20 epoche, consentendo al modello di mantenere una buona capacità di apprendimento senza incorrere in problemi di sovradattamento. Il wait decay, anche se impostato con valori bassi, ha mostrato di limitare le capacità di apprendimento del modello. Questo perché un'eccessiva riduzione del learning rate impedisce al modello di adattarsi correttamente ai dati, compromettendo la qualità della generalizzazione.

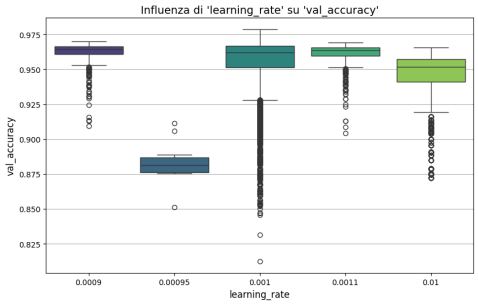


Fig. 2.51: Performance rispetto al learning rate

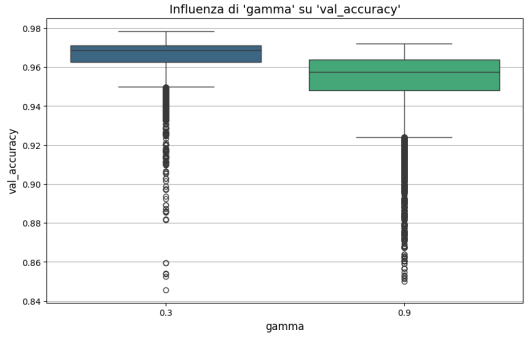


Fig. 2.52: Performance rispetto a gamma

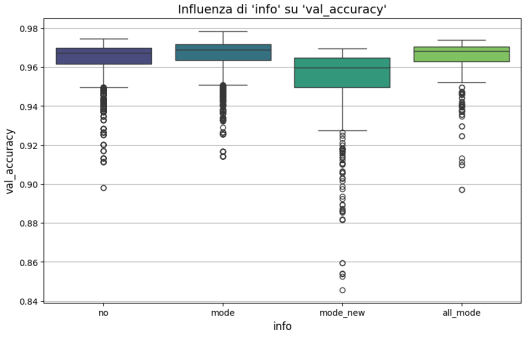


Fig. 2.53: Performance rispetto al alla sostituzione dei valori di default

Sono stati valutati diversi approcci per la gestione dei valori nulli, tra cui il riempimento dell'intera colonna con la moda e il riempimento solo delle colonne del protocollo. Il secondo metodo ha fornito le migliori prestazioni (2.53), suggerendo che un approccio più mirato alla gestione dei valori mancanti è preferibile rispetto a una strategia globale. L'analisi ha evidenziato che il Tab Transformer offre le migliori performance con il mantenimento degli outlier o la loro rimozione solo in casi estremi, un bilanciamento delle classi con almeno 20.000 istanze per classe, la riduzione della dimensionalità con LDA, la normalizzazione L1, un'architettura con 2-3 layer, l'uso di mini batch di dimensione 64, un learning rate di 0.001 con gamma 0.3 ogni 15-20 epoche e una gestione mirata dei valori nulli. Questi risultati forniscono indicazioni utili per ottimizzare ulteriormente l'addestramento e la generalizzazione del modello.

2.3.2 TabNet

Implementazione e Addestramento di TabNet

TabNet è un modello di deep learning basato su reti neurali progettato per operare con dati tabulari, che utilizza un meccanismo di attenzione sequenziale per selezionare in modo dinamico le caratteristiche più rilevanti per ogni fase dell'apprendimento.
Per l'implementazione, è stata utilizzata la libreria PyTorch TabNet, che fornisce strumenti per addestrare sia classificatori (TabNetClassifier) e permette di fare un pretraining non supervisionato su un modello TabNet (TabNetPretrainer).

Preparazione del dataset Infine anche la preparazione di TabNet ha seguito un approccio simile a quello adottato per i modelli. Poiché TabNet è un modello basato su reti neurali che utilizza un meccanismo di attenzione per selezionare le caratteristiche rilevanti, state rimosse solo le variabili categoriche con alta cardinalità, che compromettere l'accuratezza del modello. Come nei modelli precedenti, la colonna `service` è stata trasformata in una rappresentazione binaria, e gli indirizzi IP sono stati categorizzati in classi. Le porte di comunicazione sono state suddivise nei intervalli standard, mentre le variabili booleane sono state uniformate ai valori `True/False`. Inoltre, per la gestione dei valori mancanti, è stato utilizzato lo stesso approccio degli altri modelli, sostituendo i valori mancanti (individuati come –) con la moda della colonna corrispondente.
Per provare un numero maggiore di dombinazioni di iperparametri ci si è concentrati solo sulla normalizzazione e sulle due tecniche di riduzione degli outlier che avevano dato migliori performance nei precedenti modelli, ovvero `no` e `base`. Si è testato anche un solo metodo di riduzione di dimensionalità, ovvero LDA, con l'obiettivo di ridurre la complessità computazionale senza compromettere la capacità predittiva del modello.

Addestramento e tuning degli iperparametri

L'addestramento del modello è stato eseguito utilizzando il TabNetClassifier, con un approccio supervisionato. Inoltre, per migliorare la qualità dell'apprendimento, è stato sfruttato un modello TabNetPretrainer per il pre-addestramento non supervisionato, con l'obiettivo di inizializzare i pesi della rete in modo più efficace.
L'addestramento è stato condotto su cinque dataset bilanciati, derivanti dal processo di preprocessing, con un numero di campioni che è stato poi variato per analizzare quanto questo influisse sui risultati. Gli iperparametri principali ottimizzati durante il tuning includono:

- **n_d_a** (dimensione dei nodi latenti): controllano la rappresentazione delle feature e la capacità di apprendimento del modello.
- **n_steps**: determina il numero di passi decisionali, ovvero quante volte il modello può selezionare nuove feature durante l'apprendimento.
- **gamma**: bilancia l'importanza della selezione delle feature tra i vari passaggi decisionali, influenzando il comportamento del mascheramento.
- **n_independent e n_shared** (numero di unità indipendenti e condivise): regolano rispettivamente il livello di separazione e condivisione tra i diversi step del modello.
- Ottimizzatore e parametri di apprendimento, tra cui l'uso di **AdamW** e il tuning del **learning rate**.
- **epsilon**: parametro per la stabilità numerica
- **learning_rate**: velocità di apprendimento,
- **pretraining_ratio**: percentuale dei dati utilizzati per il pre-addestramento
- **momentum**: coefficiente per l'ottimizzazione, testato con valori di **0.9** e **0.99**.

Iperparametro	Valori Testati
outlier	no, base
dim_reduction	LDA
scaler	l1, l2
target_count	15000, 20000, 25000
batch_size	64, 128, 256, 512, 1024
learning_rate	0.001, 0.01, 0.1
gamma	1.0, 1.1, 1.2
patience	20
n_d_a	32, 64, 128, 512
n_shared	1, 2
n_indipendents	1, 2
n_steps	3, 5, 8, 10
epsilon	1e-15, 1e-12
pretraining_ratio	0.1, 0.5, 0.7
momentum	0.8, 0.9, 0.99
num_epochs	20, 30

Tab. 2.8: Iperparametri testati

Risultati del tuning Poiché TabNet presenta un numero elevato di iperparametri, si è scelto di limitare l'uso di tecniche di scaling e gestione degli outlier, al fine di testare un numero maggiore di combinazioni di configurazioni del modello. Basandosi sui risultati ottenuti dai modelli precedenti, si è optato per strategie mirate al bilanciamento della magnitudo delle feature, selezionando come tecniche di normalizzazione L1 e L2. Mentre per quanto riguarda il trattamento degli outlier, sono stati considerati due approcci: l'assenza di rimozione (no) e una strategia basata su soglie fisse (base). In merito alla riduzione della dimensionalità, il modello è stato testato esclusivamente con LDA (Linear Discriminant Analysis), poiché le sue caratteristiche si prestano particolarmente bene alla classificazione.

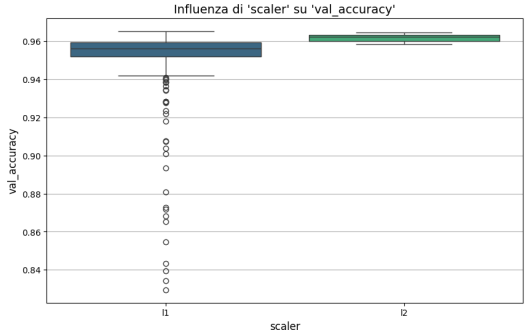


Fig. 2.54: Performance rispetto alla normalizzazione

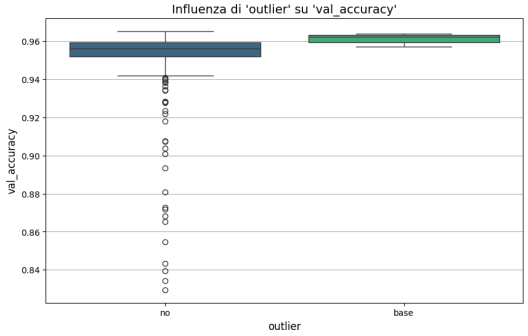


Fig. 2.55: Performance rispetto algli outlier

Come riscontrato anche negli altri modelli, la normalizzazione L1 offre prestazioni leggermente superiori rispetto alla normalizzazione L2 (2.54). Per quanto riguarda la gestione degli outlier, la loro rimozione ha determinato un lieve peggioramento delle performance, sebbene la differenza rispetto ai risultati ottenuti sul dataset originale non sia particolarmente marcata (2.55).

Per valutare l'effetto del bilanciamento delle classi, oltre al valore standard di 20.000 istanze per classe, si è scelto di testare 15.000 istanze, al fine di verificare se una riduzione del numero di esempi sintetici potesse migliorare le prestazioni del modello e parallelamente anche un aumento del numero di istanze a 25.000, per determinare se una maggiore quantità di dati, seppur generati sinteticamente o duplicati, potesse favorire l'apprendimento del modello (2.56). Tuttavia, nessuna di queste due configurazioni ha prodotto risultati sperati. I risultati evidenziano che il modello riesce a generalizzare meglio con 20.000 istanze per classe, garantendo una maggiore stabilità delle prestazioni e una riduzione della varianza tra le diverse metriche di valutazione. Pertanto, si è deciso di utilizzare questo valore come riferimento per l'analisi dei diversi iperparametri.

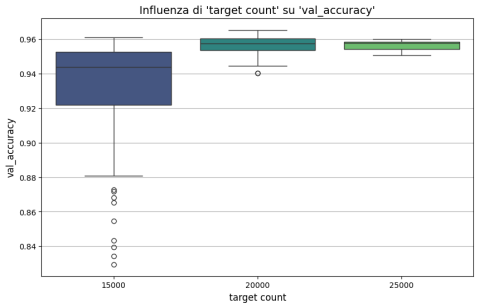


Fig. 2.56: Performance rispetto alla dimensione del bilanciamento

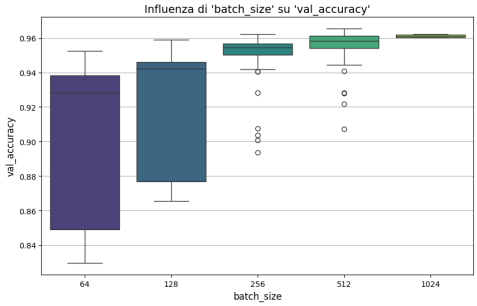


Fig. 2.57: Performance rispetto alla batch size

L'analisi degli iperparametri ha evidenziato che uno dei fattori più rilevanti per il miglioramento delle prestazioni è il batch size (2.57). Infatti, all'aumentare di tale valore, si è osservato un incremento della qualità del modello, riconducibile alla maggiore stabilità nella stima del gradiente e a una migliore capacità di generalizzazione. Tuttavia, batch size eccessivamente elevati possono ridurre la capacità del modello di adattarsi alle variazioni locali dei dati, fenomeno che potrebbe spiegare il plateau nelle prestazioni per valori molto elevati.

Anche il parametro gamma (2.58), che regola l'entità del contributo delle maschere di attenzione tra i diversi step di propagazione, ha mostrato un impatto sulle prestazioni del modello. In particolare, si è osservato che per gamma pari a 1.0 le prestazioni risultano migliori rispetto a valori più elevati. Questo suggerisce che un'influenza troppo marcata delle iterazioni precedenti può limitare la capacità del modello di adattarsi ai dati, riducendo la sua efficacia nell'estrazione di rappresentazioni discriminative.

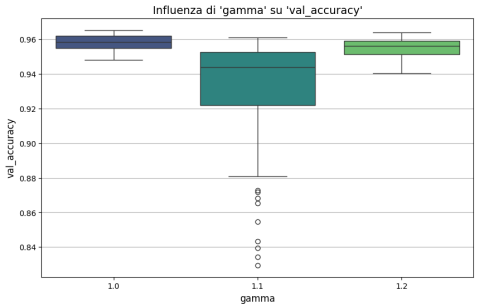


Fig. 2.58: Performance rispetto a gamma

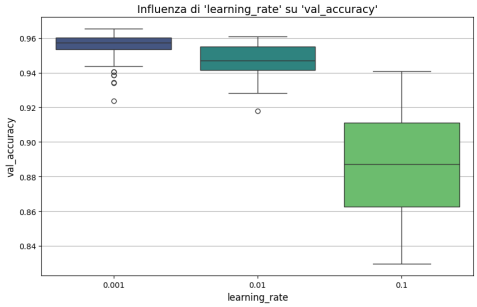


Fig. 2.59: Performance rispetto a learning rate

Un altro iperparametro che ha mostrato un impatto significativo è il learning rate (lr), che con un valore pari a 0.001 garantisce le migliori prestazioni, mentre per valori più alti, come 0.01 o 0.1, mostra un significativo peggioramento delle performance (2.59). Questo comportamento è coerente poiché TabNet essendo basato su un meccanismo di attenzione sequenziale, necessita di un apprendimento progressivo e stabile che viene compromesso da tassi di apprendimento troppo elevati.

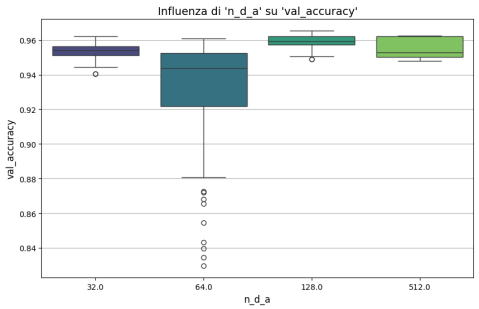


Fig. 2.60: Performance rispetto a nda

Un parametro specifico di TabNet, n_d_a (2.60), ha mostrato variazioni di performance più evidenti rispetto ad altri iperparametri. In particolare, un valore di 128 ha garantito prestazioni leggermente superiori rispetto ad altri valori testati, suggerendo che un numero adeguato di unità latenti sia determinante per la capacità del modello di apprendere rappresentazioni efficaci dei dati.

Gli altri iperparametri del modello hanno mostrato variazioni marginali nelle prestazioni, senza evidenziare un valore ottimale univoco che garantisse miglioramenti significativi. Questo potrebbe indicare che TabNet è relativamente robusto rispetto a queste variabili o che la loro influenza è minore rispetto a quella di batch size e step size.

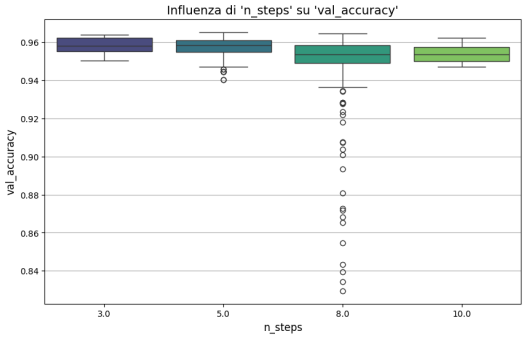


Fig. 2.61: Performance rispetto a step_size

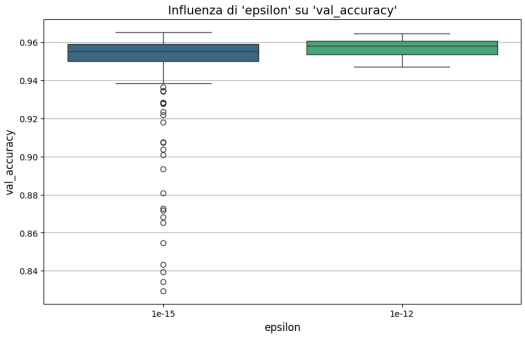


Fig. 2.62: Performance rispetto a epsilon

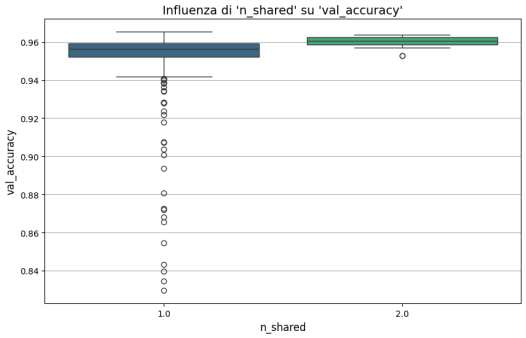


Fig. 2.63: Performance rispetto a shared

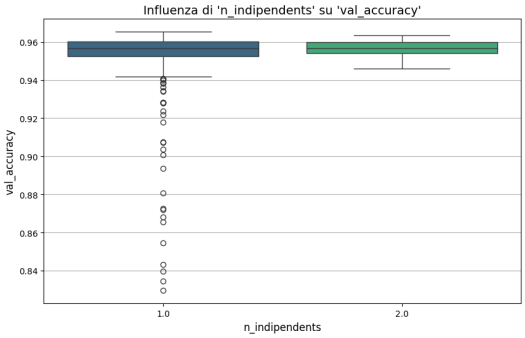


Fig. 2.64: Performance rispetto a indipendente

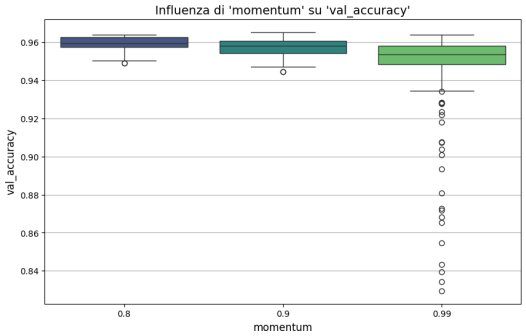


Fig. 2.65: Performance rispetto a momentum

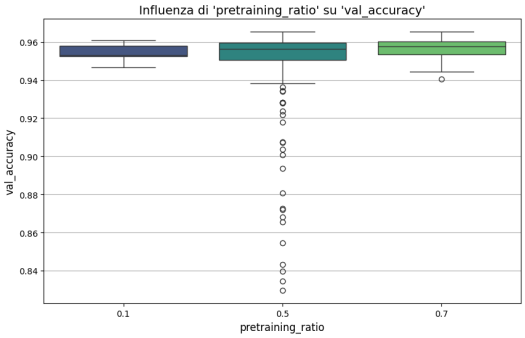


Fig. 2.66: Performance rispetto a ratio

Nonostante l'ottimizzazione degli iperparametri, le prestazioni di TabNet si sono rivelate inferiori rispetto agli altri modelli testati, con valori di accuracy che hanno raggiunto un massimo di 0.965. Nel complesso, i risultati suggeriscono che, sebbene TabNet sia un modello discreto ma che la sua efficacia non superi quella di modelli più tradizionali nel contesto specifico analizzato.

Capitolo 3

Risultati

3.1 SVM

Validation Score	Train Score	Validation Balance Accuracy	F1-Score
0.9738	0.9815	0.9641	0.9760

Tab. 3.1: Risultati della configurazione SVM: {'C': 0.5, 'kernel': 'poly', 'gamma': 0.2, 'degree': 5, 'normalizzazione': quantile, 'dim_reduction': LDA}

L'analisi delle performance del modello SVM con configurazione ottimale (`C=0.5`, `kernel=poly`, `gamma=0.2`, `degree=5`) ha evidenziato risultati molto positivi, con un'accuratezza di validazione del 97.38% e un F1-score di 97.60%. Tuttavia, un'analisi più dettagliata delle singole classi rivela alcune criticità legate alla capacità di generalizzazione e alla confusione tra specifiche categorie di attacco.

Classe	Precisione	Recall	F1-score	Supporto
Backdoor (0)	0.7938	0.9987	0.8845	1507
DDOS (1)	0.9962	0.9692	0.9825	18294
DOS (2)	0.9728	0.9954	0.9840	5420
Injection (3)	0.8504	0.9337	0.8901	1327
MITM (4)	0.3571	0.9000	0.5114	50
Normal (5)	0.9710	0.9567	0.9638	2033
Password (6)	0.9659	0.9740	0.9700	5155
Ransomware (7)	0.2488	0.9640	0.3956	111
Scanning (8)	0.9989	0.9744	0.9865	21425
XSS (9)	0.9800	0.9754	0.9777	6377
Macro avg	0.8135	0.9641	0.8546	61699
Weighted avg	0.9801	0.9738	0.9760	61699

Tab. 3.2: Metriche di classificazione per classe del modello svm

L'osservazione della matrice di confusione mostra un'elevata capacità discriminativa del modello, con precisione e recall superiori al 97% per la maggior parte delle classi. Le categorie DDoS, DoS, scanning e XSS presentano valori di F1-score superiori al 97%, suggerendo che il modello riesce a riconoscerle con grande accuratezza e con pochi errori

di classificazione. Anche la classe backdoor, pur avendo una precisione inferiore rispetto alle altre, mostra un recall vicino al 100%, indicando che il modello riesce a identificare quasi tutti i campioni appartenenti a questa categoria, anche se con una maggiore incidenza di falsi positivi.

Tuttavia, le classi meno rappresentate nel dataset, come ransomware e MITM (man-in-the-middle attack), evidenziano alcune criticità significative. La classe ransomware ha un recall molto alto (96.4%), il che significa che il modello riconosce quasi tutti i campioni reali, ma al costo di una precisione molto bassa (24.88%), segnalando un alto numero di falsi positivi. Questo fenomeno potrebbe essere attribuito alla scarsa rappresentatività della classe nei dati di addestramento, rendendo difficile per il modello apprendere caratteristiche distintive solide. Un problema simile si riscontra per la classe MITM, che presenta un recall del 90% ma una precisione di appena il 35.7%, indicando una marcata confusione con altre categorie.

Nonostante l'uso di diverse strategie di bilanciamento, il problema persiste, suggerendo che la soluzione più efficace potrebbe essere l'aumento del numero di campioni di queste classi per migliorare la capacità del modello di riconoscerle con maggiore precisione.

La matrice di confusione non normalizzata mostra una buona capacità del modello di classificare correttamente le classi più rappresentate, mentre evidenzia difficoltà con quelle meno frequenti.

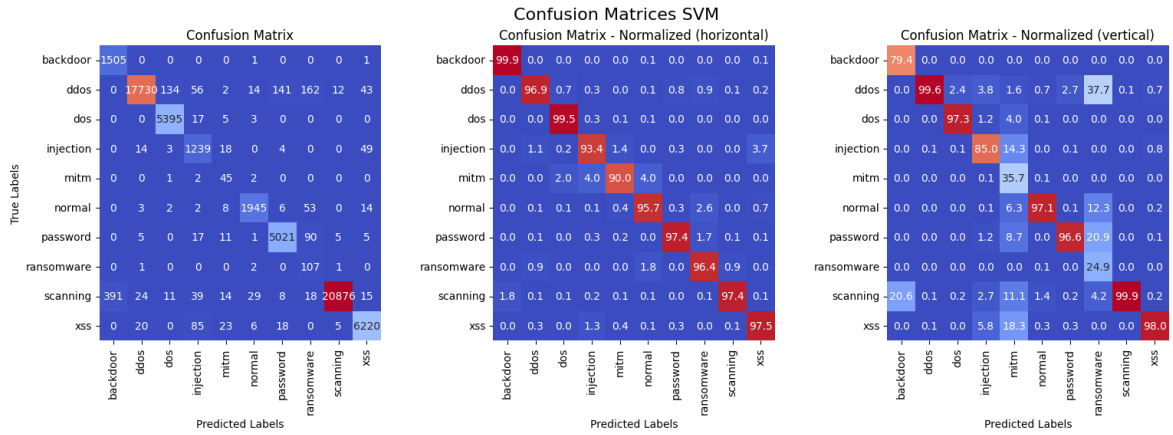


Fig. 3.1: Confusion Matrix del modello svm

L'analisi delle matrici di confusione normalizzate per riga e per colonna aiuta a comprendere meglio la distribuzione degli errori di classificazione.

- **Matrice normalizzata per riga (orizzontale)**: evidenzia un'elevata capacità di riconoscere le istanze di ciascuna classe, con errori che emergono principalmente nelle categorie meno rappresentate come ransomware e MITM.
- **Matrice normalizzata per colonna (verticale)**: mostra che alcune classi, come DDoS e scanning, vengono identificate con precisione quasi perfetta, mentre altre subiscono una maggiore contaminazione da predizioni errate, suggerendo che il modello ha difficoltà a distinguere alcune classi con caratteristiche simili.

Dunque, un'analisi comparativa tra diverse metriche evidenzia che il modello ha una buona capacità di generalizzazione, con varianza contenuta tra accuracy, balanced accuracy e F1-score. Tuttavia, la macro average F1-score (85.45%) risulta significativamente inferiore rispetto alla weighted average F1-score (97.60%), confermando che il modello è ottimizzato principalmente per le classi più numerose, mentre incontra maggiori difficoltà con le categorie meno rappresentate.

3.2 Random Forest

Validation Score	Train Score	Validation Balance Accuracy	F1-Score
0.9771	0.9884	0.9701	0.9792

Tab. 3.3: Risultati della configurazione Random Forest: {'n_estimators': 100, 'max_depth': 15, 'criterion': 'log_loss', 'ccp_alpha': 0, 'bootstrap': False, 'normalizzazione': l1, 'dim_reduction': LDA}

L'analisi delle performance del modello Random Forest evidenzia un'accuratezza di validazione pari a 97.71% e un F1-score di 97.92% con iperparametri: 'n_estimators': 100, 'max_depth': 15, 'criterion': 'log_loss', 'ccp_alpha': 0, 'bootstrap': False , valori leggermente superiori a quelli ottenuti con SVM.

Classe	Precisione	Recall	F1-score	Supporto
Backdoor (0)	0.8249	0.9973	0.9030	1507
DDOS (1)	0.9972	0.9692	0.9830	18294
DOS (2)	0.9735	0.9956	0.9844	5420
Injection (3)	0.8885	0.9608	0.9232	1327
MITM (4)	0.3485	0.9200	0.5055	50
Normal (5)	0.9698	0.9631	0.9664	2033
Password (6)	0.9700	0.9734	0.9717	5155
Ransomware (7)	0.2542	0.9550	0.4015	111
Scanning (8)	0.9991	0.9778	0.9883	21425
XSS (9)	0.9841	0.9889	0.9865	6377
Macro avg	0.8210	0.9701	0.8614	61699
Weighted avg	0.9828	0.9771	0.9792	61699

Tab. 3.4: Metriche di classificazione per classe del modello random forest

Osservando la matrice di confusione non normalizzata, emerge che le classi con il maggior numero di campioni, come DDoS, DoS, Scanning e XSS, sono classificate con un'elevata precisione, con un numero estremamente basso di errori di classificazione. Tuttavia, la classe ransomware, con soli 111 campioni di test, mostra una classificazione problematica, con 106 campioni riconosciuti correttamente, ma anche un numero elevato di falsi positivi, in particolare provenienti dalle classi scanning e ddos. Anche la classe MITM, con 50 campioni totali, ha una classificazione instabile: 46 campioni sono correttamente classificati, ma diversi falsi positivi provengono da altre classi, segnalando un problema di sovrapposizione delle caratteristiche con altre categorie.

La matrice con normalizzazione orizzontale permette di osservare la distribuzione delle previsioni rispetto alle etichette reali. La classe backdoor è classificata con un recall del 99.7%, indicando che quasi tutti i campioni reali sono correttamente riconosciuti.

Tuttavia, classi come injection e mitm mostrano livelli di recall leggermente inferiori, rispettivamente del 96.1% e 92%, segnalando una maggiore probabilità di essere confuse con altre categorie. La classe ransomware, sebbene abbia un recall del 95.5%, è spesso confusa con la classe scanning, che è predominante nel dataset e tende a inglobare alcuni esempi di altre classi. Un aspetto positivo è che le classi più numerose, come ddos, dos e scanning, mostrano un recall superiore al 96%, confermando che il modello ha una solida capacità di riconoscere gli attacchi più comuni.

La matrice di confusione non normalizzata mostra una buona capacità del modello di classificare correttamente le classi più rappresentate, mentre evidenzia difficoltà con quelle meno frequenti.

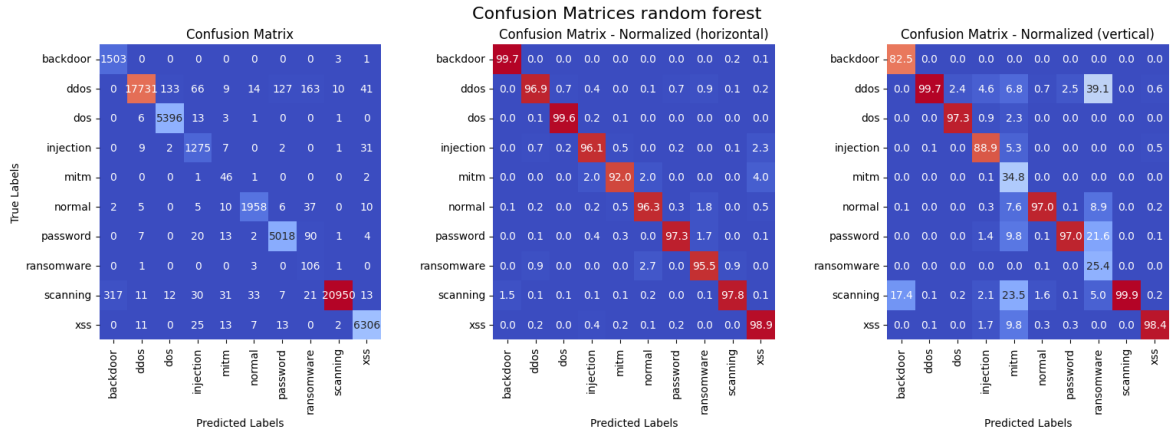


Fig. 3.2: Confusion Matrix del modello random forest

Dalla matrice normalizzata sulle colonne, nota che DDoS ha una precisione del 99.7%, il che significa che quasi tutte le istanze classificate come DDoS sono effettivamente tali. Tuttavia, la classe ransomware ha una precisione molto bassa (25.4%), evidenziando che molte istanze classificate come ransomware appartengono in realtà ad altre classi, specialmente scanning e ddos. Un problema simile si verifica per la classe MITM, che ha una precisione del 34.8%, suggerendo che il modello ha difficoltà a distinguere questa tipologia di attacco dagli altri. La classe injection, invece, ha una precisione inferiore rispetto alle altre (88.9%), con alcune istanze erroneamente classificate come XSS o dos, indicando una possibile somiglianza nelle caratteristiche dei pacchetti di rete.

Rispetto al modello SVM, il Random Forest mostra un lieve incremento dell'accuratezza e del recall per le classi più frequenti, ma introduce un numero maggiore di falsi positivi nelle classi meno rappresentate. La macro-average F1-score è 86.13%, più alta rispetto a quella dell'SVM (85.45%), il che indica una gestione leggermente migliore delle classi meno bilanciate, ma con un prezzo da pagare in termini di precisione. In particolare, le classi MITM e ransomware soffrono maggiormente nel modello RF rispetto a SVM, con un aumento della contaminazione da altre classi.

3.3 KNN

Validation Score	Train Score	Validation Balance Accuracy	F1-Score
0.9781	0.9839	0.9698	0.9792

Tab. 3.5: Risultati della configurazione KNN: {'n_neighbors': 5, 'weights': 'distance', 'metric': 'euclidean', 'normalizzazione': l1, 'dim_reduction': LDA}

L'analisi delle performance del modello **K-Nearest Neighbors (KNN)** con configurazione $k = 5$, *weights=distance*, *metric=euclidean* evidenzia un validation score del 97.81%, molto vicino alla performance ottenuta in fase di training (98.39%), suggerendo una buona generalizzazione del modello. Tuttavia, osservando la matrice di confusione e il report di classificazione, emergono criticità specifiche legate alla gestione delle classi meno rappresentate, come MITM e ransomware.

Classe	Precisione	Recall	F1-score	Supporto
Backdoor	0.8276	0.9940	0.9032	1507
DDOS	0.9959	0.9694	0.9825	18294
DOS	0.9720	0.9934	0.9826	5420
Injection	0.8585	0.9691	0.9104	1327
MITM	0.3871	0.9600	0.5517	50
Normal	0.9748	0.9685	0.9716	2033
Password	0.9611	0.9909	0.9757	5155
Ransomware	0.3929	0.8919	0.5455	111
Scanning	0.9990	0.9781	0.9884	21425
XSS	0.9852	0.9823	0.9837	6377
Macro avg	0.8354	0.9698	0.8795	61699
Weighted avg	0.9815	0.9781	0.9792	61699

Tab. 3.6: Metriche di classificazione per classe del modello knn

La matrice di confusione non normalizzata mostra una buona capacità del modello di classificare correttamente le classi più rappresentate, mentre evidenzia difficoltà con quelle meno frequenti.

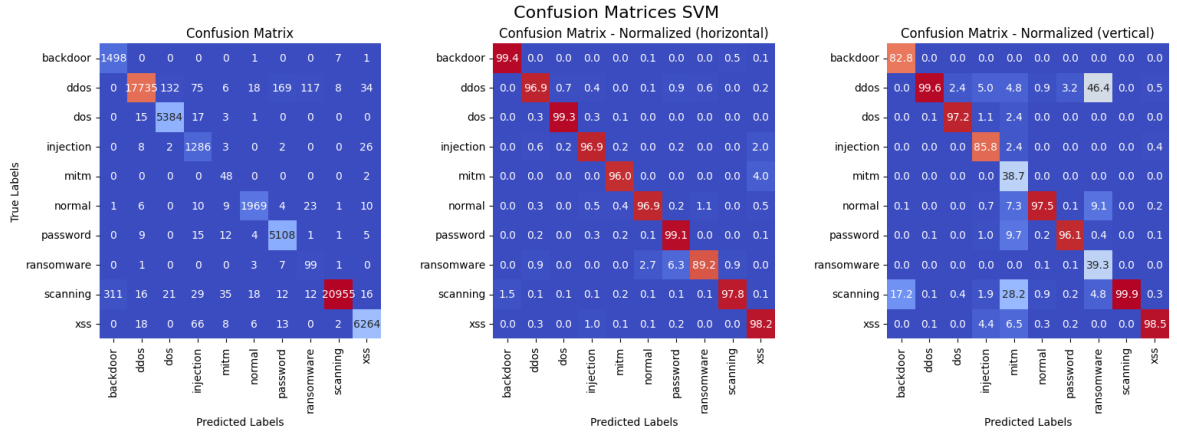


Fig. 3.3: Confusion Matrix del modello knn

Le classi DDoS, DoS, Scanning e XSS sono riconosciute con elevata precisione e recall, ottenendo un F1-score superiore al 97%. Questo risultato è dovuto alla loro ampia presenza nel dataset, che facilita l'apprendimento del modello e riduce la probabilità di errore.

Alcune classi presentano invece prestazioni intermedie. La classe Injection, ad esempio, ha un F1-score del 91.04%, caratterizzato da un recall molto elevato (96.91%) ma una precisione inferiore (85.85%), suggerendo una leggera confusione con altre categorie. Anche la classe Password mostra buone performance, con un F1-score di 97.57% e valori di precisione e recall superiori al 96%.

Le principali criticità emergono nelle classi MITM e ransomware. Nel caso di MITM, il recall è alto (96.0%), indicando che quasi tutte le istanze appartenenti a questa classe vengono riconosciute correttamente, ma la precisione è molto bassa (38.7%), il che suggerisce un elevato numero di falsi positivi. L'F1-score complessivo si attesta quindi al 55.17%, evidenziando una significativa instabilità nella classificazione. Ransomware mostra una situazione simile, con un recall dell'89.2% ma una precisione di appena 39.3%, portando a un F1-score di 54.5%. L'elevato numero di falsi positivi per queste classi è probabilmente dovuto alla loro scarsa rappresentazione nel dataset, che ne compromette la separabilità rispetto alle altre classi.

L'analisi della matrice di confusione normalizzata per riga conferma che la classe MITM presenta un recall molto elevato, con il 96% delle istanze correttamente riconosciute, mentre ransomware ha un recall dell'89.2%, pur mostrando una forte confusione con altre classi come password e scanning. Tuttavia, analizzando la matrice normalizzata per colonna, emergono criticità relative alla precisione: la classe MITM ha una precisione del 38.7%, suggerendo che molte istanze assegnate a questa categoria appartengano in realtà ad altre classi, mentre ransomware registra una precisione del 39.3%, indicando che viene spesso predetta erroneamente. Questi dati confermano che il principale limite del modello KNN è l'elevata confusione tra classi con pochi campioni, con un impatto negativo sulle metriche di precisione.

Il modello KNN raggiunge un macro-average F1-score di 87.95%, un valore comparabile con altri algoritmi. Il confronto con Random Forest evidenzia che KNN ottiene un F1-score leggermente superiore e una migliore gestione delle classi meno rappresentate.

3.4 Reti

Validation Score	Train Score	Validation Loss	Train Loss
0.9791	0.9857	0.0964	0.0508

Balanced Accuratezza	F1-Score
0.9642	0.9803

Tab. 3.7: Risultati della configurazione: {'weight_decay': 0, 'batch_size': 128, 'hidden_size': 256, 'batch_norm': False, 'dropout': 0.1, 'depth': 3, 'learning_rate': 0.001, 'gamma': 0.3, 'step_size': 20, 'num_epochs': 250, 'normalizzazione': l1, 'dim_reduction': LDA}

L'analisi delle performance delle reti feedforward (FF) evidenzia risultati di alto livello, con un'accuratezza bilanciata del 96.42% e un F1-score medio pesato di 98.03%. Questi

valori indicano una capacità molto elevata del modello nel distinguere le diverse classi, confermando la bontà della configurazione adottata. Le classi con il maggior numero di campioni, come DDoS, Scanning e XSS, raggiungono valori di precisione, recall e F1-score estremamente alti, dimostrando che il modello è in grado di identificare con grande precisione e affidabilità questi specifici attacchi. In particolare, la classe Scanning ottiene un F1-score di 98.80%, con una precisione prossima al 100%, il che suggerisce una quasi perfetta capacità di riconoscere questo tipo di comportamento anomalo nella rete.

Classe	**Precisione**	**Recall**	**F1-score**	**Supporto**
Backdoor (0)	0.7959	0.9987	0.8858	1507
DDOS (1)	0.9976	0.9715	0.9844	18294
DOS (2)	0.9752	0.9948	0.9849	5420
Injection (3)	0.9177	0.9586	0.9377	1327
MITM (4)	0.4945	0.9000	0.6383	50
Normal (5)	0.9768	0.9547	0.9657	2033
Password (6)	0.9599	0.9938	0.9766	5155
Ransomware (7)	0.3390	0.9009	0.4926	111
Scanning (8)	0.9993	0.9770	0.9880	21425
XSS (9)	0.9852	0.9923	0.9888	6377
Macro avg	0.8441	0.9642	0.8843	61699
Weighted avg	0.9829	0.9791	0.9803	61699

Tab. 3.8: Report di classificazione per ciascuna classe.

Tuttavia, analizzando più nel dettaglio le matrici di confusione, emergono alcune difficoltà nella classificazione delle classi meno rappresentate, come MITM e Ransomware. La classe MITM mostra un recall del 90%, il che significa che la maggior parte delle istanze di questo tipo vengono correttamente identificate, ma al tempo stesso presenta una precisione più bassa, pari al 49.45%. Questo valore suggerisce che molte predizioni errate sono state assegnate a questa classe, evidenziando la presenza di un numero elevato di falsi positivi. Anche la classe Ransomware evidenzia un comportamento simile, con un recall molto elevato, pari al 90.09%, ma una precisione sensibilmente inferiore, attestandosi al 33.90%. Questo fenomeno suggerisce che il modello tende ad assegnare erroneamente a questa classe istanze appartenenti ad altre categorie, riducendo l'affidabilità della predizione.

Osservando la matrice di confusione normalizzata, si può notare come alcune classi siano classificate quasi perfettamente. La classe Backdoor, ad esempio, ottiene un recall pari al 99.87%, indicando che praticamente tutte le istanze appartenenti a questa categoria sono state correttamente identificate. Anche la classe Password si distingue per un comportamento simile, con un recall del 99.38% e un F1-score molto elevato. La classe DOS, allo stesso modo, raggiunge un recall del 99.48% e un F1-score del 98.49%, dimostrando la capacità del modello di riconoscere con precisione questo tipo di attacco. Tuttavia, analizzando la classe Normal, si osserva una lieve flessione nelle prestazioni: la precisione rimane alta, attestandosi al 97.68%, ma il recall scende leggermente al 95.47%. Questo dato suggerisce che alcune istanze di traffico normale vengono erroneamente classificate come attacchi, riducendo la capacità del modello di distinguere in modo netto il comportamento lecito da quello malevolo.

Dal punto di vista dell'addestramento, la rete FF ha mostrato un'ottima capacità di ap-

prendimento, grazie alla profondità del modello e all'utilizzo di dropout, che ha permesso di evitare fenomeni di overfitting. La configurazione adottata ha hottenuno i risultato migliori all'eopoca 78, con una perdita di validazione finale di 0.0964, mentre la perdita di addestramento si è stabilizzata a 0.0508. Questa differenza contenuta tra le due perdite suggerisce che il modello ha generalizzato bene, senza incorrere in problemi di overfitting o underfitting.

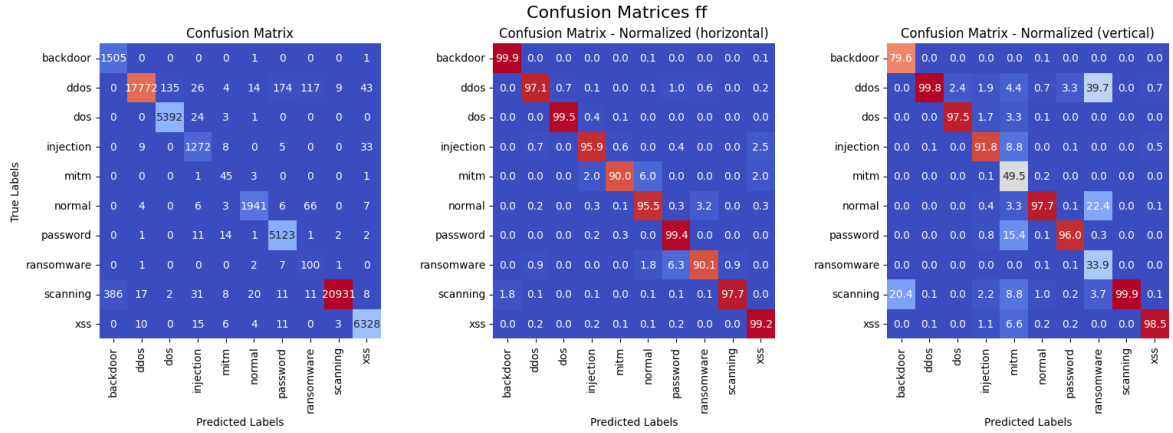


Fig. 3.4: Confusion Matrix del modello delle reti

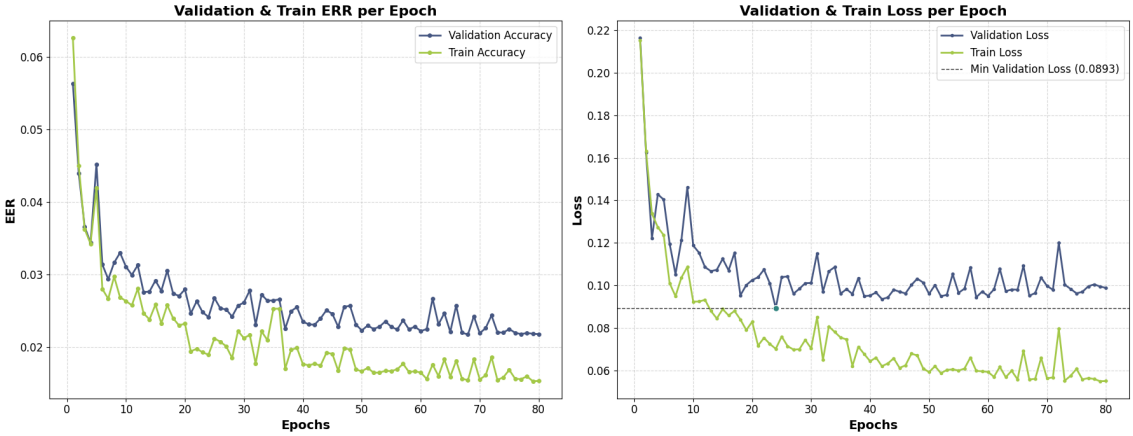


Fig. 3.5: Andamento dell'accuracy e della loss con gamma e lr al variare dell'epoca

Infatti, l'analisi delle performance mostra un rapido miglioramento iniziale, con l'errore (ERR) che cala velocemente nelle prime epoche prima di stabilizzarsi. Tuttavia, dopo alcune epoche, il divario tra training e validation comincia ad amuentare suggerendo che il modello tenda verso il fenomeno dell'overfitting, con il modello che continua a migliorare sui dati di training senza un corrispondente beneficio sulla generalizzazione. La curva della loss conferma questa tendenza: mentre la loss di training si riduce progressivamente, quella di validazione oscilla dopo una fase iniziale di discesa, indicando che il modello stia iniziando a memorizzare piuttosto che apprendere pattern generali.

Nel complesso, i risultati ottenuti indicano che la configurazione utilizzata per la rete FF è altamente performante e adatta al compito di classificazione del traffico di rete. Tuttavia, le difficoltà incontrate nella classificazione di classi meno rappresentate suggeriscono che potrebbero essere necessari alcuni accorgimenti per migliorare ulteriormente

le prestazioni.

3.5 TabTrasform

Validation Score	Train Score	Validation Loss	Train Loss
0.9785	0.9843	0.08885	0.05604

Balanced Accuratezza	F1-Score
0.9675	0.9798

Tab. 3.9: Risultati della configurazione: {'weight_decay': 0, 'batch_size': 64, 'hidden_size': 256, 'dim_embedding': 64, 'batch_norm': False, 'dropout': 0.1, 'depth': 3, 'learning_rate': 0.001, 'gamma': 0.3, 'step_size': 20, 'num_epochs': 250, 'normalizzazione': l1, 'dim_reduction': LDA}

L'analisi delle prestazioni del modello TabTransformer conferma la sua capacità di classificazione efficace, soprattutto per le classi più rappresentate, ma evidenzia ancora una volta le difficoltà nel distinguere correttamente le classi meno frequenti. Il modello ha raggiunto un F1-score medio pesato del 97.98% e un'accuratezza complessiva del 97.85%, valori che si avvicinano a quelli delle reti neurali, sebbene con alcune differenze nelle prestazioni per le singole classi.

Le classi con maggiore rappresentatività nel dataset, come DDoS, Scanning e XSS, hanno ottenuto F1-score molto elevati, dimostrando che il modello è estremamente efficace nel riconoscere questi tipi di attacchi con un numero minimo di falsi positivi e falsi negativi. Tuttavia, il problema della classificazione delle classi meno rappresentate persiste. La classe MITM, per esempio, ottiene un F1-score del 61.04%, con una recall molto alta del 94% ma una precisione molto bassa del 45.19%, segnalando un'elevata quantità di falsi positivi. Questo significa che molte istanze appartenenti ad altre classi vengono erroneamente classificate come MITM, riducendo l'affidabilità del modello per questo tipo di attacco. Una situazione simile si verifica con la classe Ransomware, che ottiene una recall del 90.09% ma una precisione del 33.33%, con un F1-score finale di 48.66%. Ciò indica che, sebbene il modello riesca a identificare quasi tutti i ransomware presenti nei dati, classifica erroneamente molte altre istanze come tali, portando a una frequenza elevata di falsi positivi e riducendo la precisione complessiva.
Un aspetto interessante riguarda la classe Injection, che nel caso del TabTransformer mostra un leggero miglioramento rispetto agli altri modelli, con un F1-score del 94.13%, una precisione del 92.88% e una recall del 95.40%. Questo potrebbe essere dovuto alla capacità del modello di sfruttare al meglio i meccanismi di attenzione per catturare pattern più complessi nei dati tabulari, un punto di forza delle architetture transformer rispetto alle reti completamente connesse.

Classe	Precisione	Recall	F1-score	Supporto
Backdoor (0)	0.7959	0.9987	0.8858	1507
DDOS (1)	0.9976	0.9707	0.9840	18294
DOS (2)	0.9720	0.9972	0.9844	5420
Injection (3)	0.9288	0.9540	0.9413	1327
MITM (4)	0.4520	0.9400	0.6104	50
Normal (5)	0.9690	0.9528	0.9608	2033
Password (6)	0.9590	0.9934	0.9759	5155
Ransomware (7)	0.3333	0.9009	0.4866	111
Scanning (8)	0.9993	0.9760	0.9875	21425
XSS (9)	0.9853	0.9914	0.9884	6377
Macro avg	0.8392	0.9675	0.8805	61699
Weighted avg	0.9825	0.9785	0.9798	61699

Tab. 3.10: Report di classificazione per ciascuna classe.

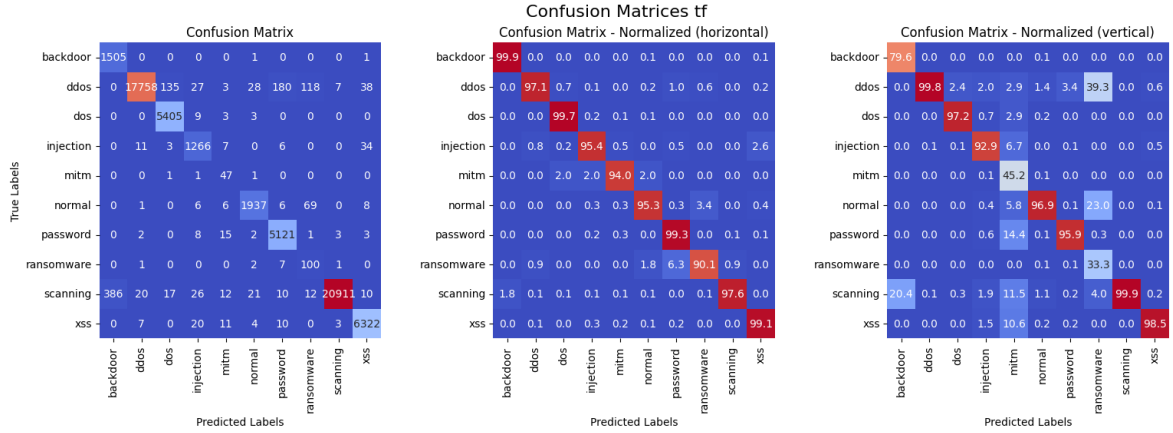


Fig. 3.6: Confusion Matrix del modello basato su TabTrasform

Analizzando la matrice di confusione, emerge che il modello continua a classificare con estrema precisione le classi dominanti, ma mostra incertezze significative nel riconoscimento delle classi meno rappresentate. La matrice normalizzata verticalmente conferma che una quota consistente delle istanze MITM viene erroneamente classificate come Normal o Injection, mentre una percentuale significativa delle istanze di Ransomware viene scambiata per altre classi, segnalando una necessità di miglioramento nella discriminazione tra le categorie meno frequenti.

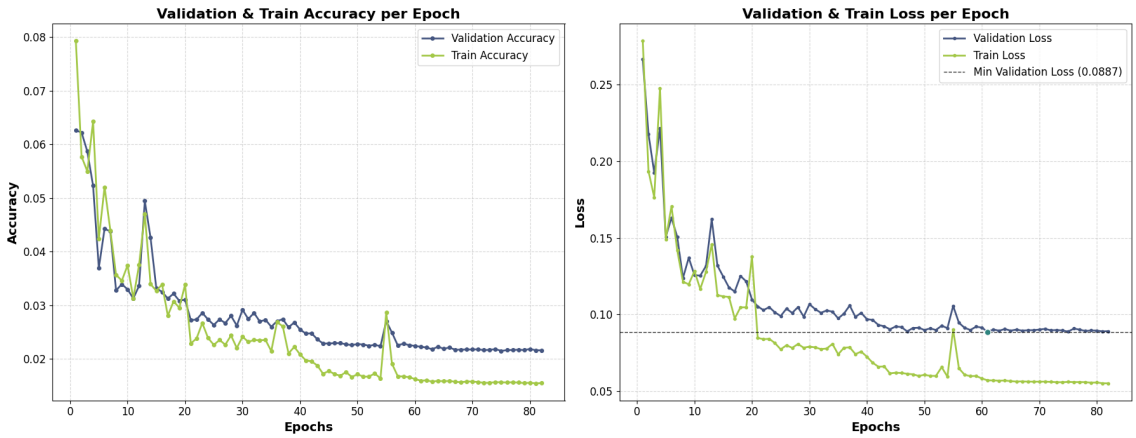


Fig. 3.7: Andamento dell'accuracy e della loss con gamma e lr al variare dell'epoca

L'analisi grafica evidenzia che l'Equal Error Rate (ERR) decresce rapidamente nelle prime epoche e si stabilizza senza oscillazioni significative, suggerendo un buon livello di generalizzazione. Anche la loss segue un comportamento regolare, con una progressiva diminuzione della train loss e una validation loss che non mostra elevata varianza, segnalando che il modello non soffre di overfitting evidente e generalizza bene sui dati di validazione.

Nel complesso, il TabTransformer si dimostra un modello robusto e affidabile, con ottime prestazioni sulle classi più frequenti e un buon equilibrio tra capacità di apprendimento e generalizzazione. Tuttavia, rimane spazio per miglioramenti nella gestione delle classi meno rappresentate, ad esempio con strategie di riequilibrio dei dati o di raffinamento della regolarizzazione.

3.6 Tabnet

Accuracy	Loss	Validation Balance Accuracy	F1-Score
0.9652	0.7262	0.9417	0.9678

Tab. 3.11: Risultati della configurazione: {'num_epochs': 30, 'batch_size': 512, 'patience': 20, 'n_d_a': 128, 'n_shared': 1, 'n_indipendents': 1, 'n_steps': 5, 'gamma': 1.0, 'epsilon': 1e-15, 'learning_rate': 0.001, 'pretraining_ratio': 0.7, 'momentum': 0.9, 'normalizzazione': l1, 'dim_reduction': LDA}

Il modello TabNet è stato addestrato con una configurazione che prevede 30 epoche, un batch size di 512 e un criterio di early stopping con una pazienza di 20 epoche. I parametri principali includono 128 unità per le trasformazioni delle feature e l'attenzione, un singolo livello condiviso e indipendente, cinque passi decisionali e un learning rate di 0.001. Il modello è stato salvato alla ventiseiesima epoca, raggiungendo una migliore accuratezza di validazione del 96.52%.

Dall'analisi della matrice di confusione standard, emerge che il modello è in grado di classificare correttamente alcune classi con elevata precisione, in particolare DDoS, DoS, Scanning e XSS, che presentano un numero molto ridotto di errori. Questo in parte è dovuto al numero maggiore di campioni che presentano queste classi nel dataset: il modello classifica correttamente 17.625 istanze su 18.294 per DDoS, con un numero limitato di

errori distribuiti principalmente tra le classi DoS (133), Injection (68) e Normal. Analogamente, la classe Scanning, che conta 21.425 istanze, è stata identificata correttamente in 20.724 casi, con un margine di errore molto contenuto rispetto alle altre classi.

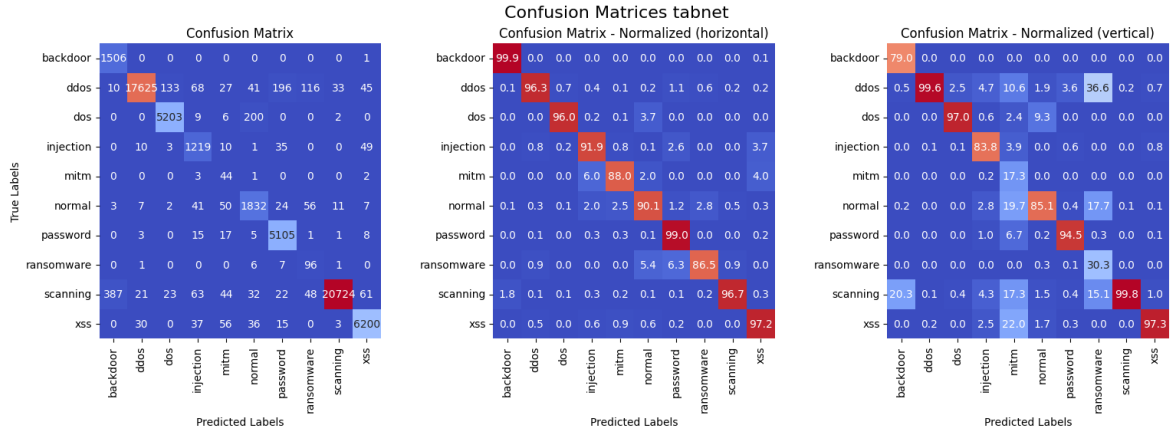


Fig. 3.8: Confusion Matrix del modello basato su TabNet

Le prestazioni globali del modello risultano ancora più evidenti dall'analisi della matrice normalizzata orizzontalmente, che riporta le percentuali di classificazione corretta per ogni classe. La classe Backdoor raggiunge una precisione del 99.9%, con pochissime predizioni errate. DDoS mantiene un'elevata precisione del 96.3%, con errori distribuiti principalmente tra DoS, Normal e Injection. La classe DoS registra un'accuratezza del 96.0%, con errori limitati verso Injection e Normal. Tuttavia, alcune categorie risultano più difficili da classificare. La classe Injection, pur avendo una buona precisione del 91.9%, presenta una dispersione di errori verso MITM e altre classi. La categoria MITM, in particolare, evidenzia difficoltà di classificazione, con una precisione dell'88.0% e un 6% delle istanze erroneamente attribuite a Normal e un 4% a XSS. Analogamente, la classe Normal è riconosciuta correttamente nel 90.1% dei casi, con il restante 9.9% distribuito tra MITM e Scanning. La classe Password si distingue per un'accuratezza molto elevata (99.0%), con un numero trascurabile di errori, mentre la classe Ransomware mostra alcune criticità, essendo classificata correttamente solo nell'86.5% dei casi, con un 5.4% di errori verso Normal e un 6.3% verso Password. Scanning si conferma tra le classi meglio classificate, con una precisione del 96.7%, mentre XSS raggiunge il valore più elevato di accuratezza con il 97.2% di predizioni corrette.

La matrice normalizzata verticalmente rivela informazioni cruciali sugli errori di classificazione rispetto alla distribuzione reale delle classi. In particolare, MITM ha una recall molto bassa, pari solo al 17.3%, il che significa che molte istanze reali di questa classe vengono erroneamente classificate come Normal o altre categorie. Anche Ransomware soffre di una recall ridotta, pari al 30.3%, evidenziando la difficoltà del modello nel riconoscere correttamente questa tipologia di attacco. DDoS, invece, si distingue per una recall estremamente alta, raggiungendo il 99.6%, con pochissimi falsi negativi. Le classi Scanning e XSS ottengono risultati eccellenti, con una recall superiore al 99%, confermando l'efficacia del modello nel rilevare queste minacce.

Dal report di classificazione emergono alcuni aspetti chiave. Si ottiene la conferma che classi più rappresentate, come DDoS, Scanning e XSS, hanno F1-score superiori al

96%, dimostrando che il modello è altamente efficace nella loro classificazione. Tuttavia, MITM e Ransomware mostrano prestazioni significativamente più deboli, con F1-score rispettivamente di 28.9% e 44.8%, a causa di un alto numero di falsi positivi e falsi negativi. Il valore medio del macro F1-score è pari a 82.38%, mentre l'accuratezza bilanciata in validazione raggiunge il 94.17%, a conferma del fatto che il modello è generalmente robusto, pur mostrando difficoltà con alcune classi meno rappresentate. Anche la classe Injection presenta F1-score di 0.8763 , inferiore rispetto alle altre classi, con un recall relativamente alto (0.9186) ma una precisione più bassa (0.8378), indicando nuovamente un problema di falsi positivi.
Questo evidenzia nuovamente l'importanza della qualità e della distribuzione dei dati nella fase di addestramento. Sebbene le tecniche di preprocessing possano contribuire a migliorare le prestazioni del modello, in assenza di un numero adeguato di dati esse risultano insufficienti per garantire un'ottimale capacità di generalizzazione.

Classe	**Precisione**	**Recall**	**F1-score**	**Supporto**
Backdoor (0)	0.7901	0.9993	0.8825	1507
DDOS (1)	0.9959	0.9634	0.9794	18294
DOS (2)	0.9700	0.9600	0.9649	5420
Injection (3)	0.8378	0.9186	0.8763	1327
MITM (4)	0.1732	0.8800	0.2895	50
Normal (5)	0.8505	0.9011	0.8751	2033
Password (6)	0.9447	0.9903	0.9669	5155
Ransomware (7)	0.3028	0.8649	0.4486	111
Scanning (8)	0.9975	0.9673	0.9822	21425
XSS (9)	0.9729	0.9722	0.9725	6377
Macro avg	0.7836	0.9417	0.8238	61699
Weighted avg	0.9724	0.9652	0.9678	61699

Tab. 3.12: Metriche di classificazione per ogni classe.

Infine, sebbene il modello presenti prestazioni leggermente inferiori rispetto agli altri analizzati, soprattutto per le classi meno rappresentate, esso dimostra comunque buone performance complessive.