SAPIENZA
UNIVERSITÀ DI ROMA

Intrusion Detection System basato su Ensemble di Modelli di Machine Learning

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica.
Laurea Triennale in Informatica

Luca Brunetti
Matricola 1967993

Relatore
Emiliano Casalicchio

Anno Accademico 2024-2025

Sommario

Negli ultimi anni, data la crescente dipendenza dalle tecnologie, gli attacchi informatici rappresentano una minaccia costante per la sicurezza delle reti e dei sistemi. In questo contesto, gli **Intrusion Detection Systems** (IDS), svolgono un ruolo cruciale nell'analisi del traffico di rete, individuando comportamenti anomali e segnali precoci di possibili intrusioni. Per adattarsi al continuo evolversi della complessità degli attacchi, numerose tecniche che sfruttano algoritmi di **Machine Learning** e **Deep Learning** sono state implementate.
L'oggetto di questa tesi è lo studio dei vari approcci e metodologie utilizzate dagli IDS per determinare se è in atto un'intrusione. Successivamente, viene proposto un ensemble di modelli di machine learning, progettato per combinare i punti di forza dei diversi approcci. Tale architettura permette di rilevare e classificare, in modo efficace, sia attacchi noti che minacce emergenti. I risultati ottenuti, confermano il potenziale dell'approccio adottato per migliorare la sicurezza delle infrastrutture informatiche, offrendo una soluzione scalabile e robusta per il rilevamento delle minacce in ambienti reali.

Indice

Capitolo 1

Introduzione

Uno dei problemi più significativi nella sicurezza informatica moderna riguarda la capacità di individuare tempestivamente minacce e anomalie all'interno di sistemi e reti. In questo contesto, gli **Intrusion Detection Systems** assumono un ruolo centrale, offrendo una prima linea di difesa contro attacchi informatici e accessi non autorizzati. Un sistema di rilevamento delle intrusioni è quindi essenziale per identificare e prevenire attività dannose, garantendo così la protezione delle infrastrutture digitali e dei dati sensibili.
Numerosi *analizzatori* negli IDS utilizzano modelli di Machine Learning per rilevare le intrusioni efficacemente, sfruttando algoritmi capaci di separare e classificare il traffico anomalo da quello legittimo. L'obiettivo di questa tesi è l'implementazione di un architettura, basata su questi algoritmi, in grado di determinare la natura del traffico di rete combinando i vantaggi delle diverse politiche di rilevazione.
La seguente relazione è strutturata come segue:

- Il Capitolo 2 fornisce un background teorico sugli IDS, la loro classificazione e la descrizione dei due approcci principali nella determinazione delle anomalie.
- Il Capitolo 3 fornisce un'introduzione teorica ai modelli di apprendimento automatico, e alle loro configurazioni, utilizzati nell'architettura di questo studio. Vengono poi descritti il processo di addestramento e l'implementazione dell'architettura proposta.
- Il Capitolo 4 presenta e analizza i risultati sperimentali ottenuti.

Capitolo 2

Background Intrusion Detection Systems

Un Intrusion Detection System è un dispositivo hardware, o un'applicazione software, progettato per monitorare e analizzare in tempo reale il traffico di rete e le attività di sistema. Il suo obiettivo è quello di identificare tempestivamente tentativi di accesso indesiderati alle risorse, rilevando comportamenti anomali che potrebbero indicare attacchi informatici. In caso di rilevamento di anomalie, l'IDS lancia allarmi e genera registri dettagliati, fornendo dati preziosi per ulteriori analisi e per il coordinamento di interventi di sicurezza.

Un IDS è formato da cinque componenti:

- **Sensori:** posizionati in punti strategici della rete o del sistema, hanno il compito di raccogliere informazioni.
- **Analizzatori:** ricevono in input i dati raccolti dai sensori, hanno lo scopo di determinare se è in atto, o se è avvenuta, un'intrusione nel sistema.
- **Interfaccia grafica:** permette ad un utente di osservare e controllare il comportamento del sistema.
- **Database:** viene utilizzato per memorizzare i dati raccolti dai sensori, incluso traffico di rete, logs di sistema e altri dati relativi alla sicurezza.
- **Alert System:** responsabile della generazione e della gestione degli avvisi quando il sistema rileva attività sospette o potenziali intrusioni. Un avviso in genere include informazioni sul tipo di attacco, il sistema colpito e sulla gravità dell'attacco.

Gli IDS possono essere categorizzati in base alla loro posizione nell'infrastruttura:

2.0.1 Network-based IDS - NIDS

Un NIDS ha il compito di monitorare il traffico di rete, analizzando i protocolli e il contenuto dei pacchetti, per identificare attività malevole. Vengono posizionati

in punti strategici come: nella *DMZ* per monitorare le comunicazioni con i servizi aperti agli estranei; nella periferia della rete per sorvegliare il traffico in entrata ed in uscita di una rete aziendale; dietro i firewall interni utili per monitorare attacchi originati dall'interno della rete. I NIDS possono identificare attacchi basati sulla rete come attacchi Denial-of-service (DoS), scansioni alle rete o alle porte (probe) ed altre attività malevole.

2.0.2 Host-based IDS - HIDS

Un *HIDS* ha il compito di monitorare le attività che avvengono all'interno di un singolo host al fine di individuare azioni sospette. Poichè un HIDS risiede direttamente all'interno delll'host, esso è in grado di analizzare i logs, i processi in esecuzione, le modifiche al file system e altre attività potenzialmente anomale. Di conseguenza, sono in grado di identificare malware, violazioni all'integrità dei file, modifiche non autorizzate alle configurazioni di sistema e altre attività malevole mirate alla compromissione dell'host.

2.1 Tecniche di intrusion detection

Il comportamento di un attaccante differisce significativamente da quello di un utente legittimo, rendendo così possibile l'identificazione di attività malevole attraverso varie tecniche e metodologie:

2.1.1 Signature-based detection

Gli IDS basati su firme utilizzano un database contenente pattern di attacchi conosciuti che vengono confrontati con il traffico di rete o le attività di sistema per identificare eventuali corrispondenze. Dato che questi IDS sono facilmente implementabili e molto accurati, sono in grado di generare pochi falsi negativi. Tuttavia è essenziale che il database delle firme venga costantemente aggiornato per includere nuove minacce. Lo svantaggio principale degli IDS signature-based è la loro vulnerabilità nei confronti di nuovi attacchi (attacchi **zero-day**) o varianti di minacce esistenti poiché si basano su firme predefinite che non includono minacce sconosciute.

2.1.2 Anomaly Detection

Il rilevamento delle anomalie è un approccio basato sull'identificazione di comportamenti che si discostano da quelli dell'utente comune. Inizialmente il sistema apprende come l'utente legittimo si comporta attraverso la raccolta e l'analisi dei dati provenienti da diversi sensori. Questo processo di apprendimento può essere continuo, consentendo al sistema di adattarsi alle variazioni nel comportamento dell'utente nel tempo e di affinare progressivamente i criteri di rilevamento.
Una volta creato questo modello, il comportamento dell'utente osservato viene confrontato per determinare se sia legittimo o anomalo attraverso varie tecniche. Tra le più comuni abbiamo:

- **Statistiche:** analisi del comportamento osservato attraverso parametri quantitativi (come media, deviazione standard, percentili, ecc. . .) per individuare variazioni significative rispetto al profilo tipico.
- **Knowledge-based:** questo approccio genera delle regole basandosi sul modello di comportamento legittimo per poi generare allarmi se il comportamento corrente viola quest'ultime.
- **Machine-learning:** i modelli di machine learning riescono a determinare automaticamente quando i dati correnti si discostano dal comportamento legittimo dato in allenamento. Questo metodo è quindi molto robusto. I modelli maggiormente utilizzati nell'**Anomaly Detection** sono le Support Vectore Machine (SVMs), i *Decision Trees* (DT), gli *Hidden Markov Models* (HMM) e il *Clustering*.

Il grande vantaggio di questo approccio è la sua robustezza rispetto agli attacchi zero-day, poiché non si basa esclusivamente sulla conoscenza di pattern conosciuti, ma piuttosto sulla capacità di individuare comportamenti devianti rispetto ad un modello di riferimento legittimo. Tuttavia ha come svantaggio l'elevata presenza di falsi positivi dati dalle minime variazioni del comportamento dell'utente, come mostra la Figura 2.1, questo rende cruciale il processo di continuo aggiornamento del modello.

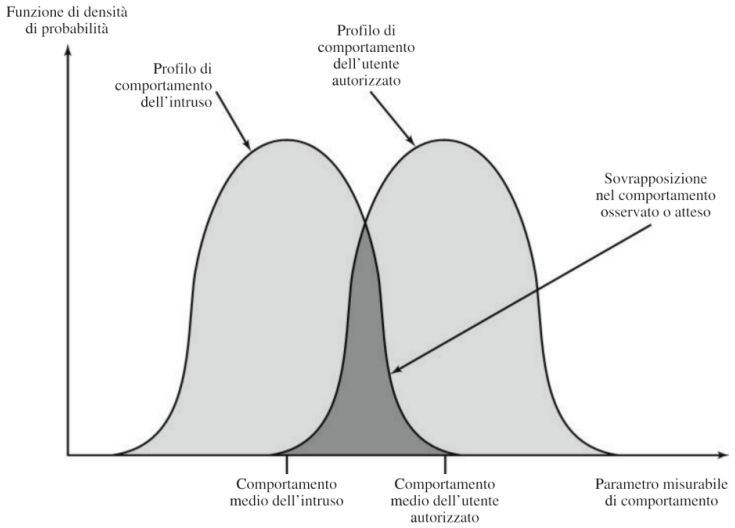


Figura 2.1. Profili di comportamento degli intrusi e degli utenti autorizzati

2.2 Network-based Intrusion Detection

Nei NIDS vengono utilizzati entrambe gli approcci signature-based ed anomaly detection per analizzare il traffico di rete dato in output dai sensori. Questi ultimi sono schierati in due modi:

- **Sensori attivi:** vengono inseriti in un segmento della rete così che il traffico ci passi attraverso, tipicamente vengono vengono combinati con altri dispositivi di rete come firewall e switch. Il vantaggio dei sensori attivi è la loro capacità di bloccare gli attacchi in tempo reale quando questi sono rilevati.

- **Sensori passivi:** monitora una copia del traffico di rete, quindi raccoglie informazioni che successivamente verranno analizzate, senza intervenire sul traffico corrente. Questo li rende maggiormente efficienti rispetto ai sensori attivi che analizzando in tempo reale possono introdurre maggiori latenze.

I sensori sono implementati e monitorano una rete cablata o wireless. Comunemente nelle reti wireless vengono implementati, dentro un wireless **Access Point** (AP), sensori attivi.

2.2.1 NIDS basati su signature detection

I NIDS signature-based confrontano i pacchetti e i flussi dati con un database di firme predefinite. Quando il traffico corrisponde ad una delle firme presenti, il NIDS genera un alert e interviene automaticamente per bloccare l'attacco. La signature detection è particolarmente efficace nel rilevare i seguenti tipi di attacchi[4]:

- **SQL Injection:** ricercando nelle richieste HTTP pattern di caratteri tipici di questo attacco come "UNION SELECT", "DROP TABLE" ecc...

- **Cross-Site Scripting:** il NIDS controlla il payload alla ricerca di pattern che includono tag HTML o script.

- **Directory Traversal:** ricerca all'interno dei percorsi richiesti sequenze come "../" che indicano tentativi di uscire dalla directory autorizzata per accedere a file o directory riservate.

- **Command Injection:** il NIDS identifica le stringhe che contengono caratteri come "&&","|" che potrebbero essere utilizzati per concatenare comandi di sistema.

- **Probe:** il NIDS analizza traffico UDP e TCP per individuare sequenze anomale di connessioni in rapida successione che mirano a sondare un insieme di porte per identificare quelle aperte o vulnerabili.

- **IP Spoofing:** analizza IPv4, IPv6, ICMP e IGMP per verificarne la coerenza e identificare discrepanze.

2.2.2 NIDS basati su Anomaly Detection

I NIDS anomaly-based costruiscono un modello di comportamento legittimo monitorando il traffico di rete, per poi confrontarlo con il traffico corrente. Gli attacchi più adatti all'anomaly detection sono:

- **Denial-of-service:** viene rilevato un incremento significativo del volume dei pacchetti nel traffico o una successione di tentativi di connessione. Questo genera una deviazione statistica facilmente riconoscibile.
- **Scanning:** il NIDS rileva sequenze di pacchetti diverse dal normale a livello applicativo, di trasporto e di rete.
- **Worms:** alcuni worm si propagano velocemente quindi utilizzando più larghezza di banda del solito. In caso di Botnet, i worm vengono rilevati poichè vari host comunicano tra di loro quando tipicamente non lo fanno.

Capitolo 3

Descrizione del problema

Gli analizzatori nei *Network Intrusion Detection Systems* svolgono il ruolo cruciale di determinare se il traffico osservato è legittimo o se indica un'intrusione avvenuta. Negli ultimi anni numerosi NIDS hanno implementato approcci di Machine e Deep Learning per rilevare anomalie; tuttavia queste tecniche non riescono ad individuare tutte le intrusioni con successo [6]. La rilevazione signature-based può essere eseguita utilizzando algoritmi di machine learning supervisionati come le *Back Propagation Artificial Neural Network (BP-ANN)* e le *Multi-Class Support Vector Machine (SVM)* in grado di rilevare anomalie con un basso numero di falsi negativi. I NIDS anomaly-based utilizzano algoritmi di machine learning supervisionati e non, in grado di distinguere il traffico normale da quello anomalo come le *Support Vector Machine (SVM)* e i *Decision Trees (DT)*. Lo svantaggio principale dell'intrusion detection basato su firme, oltre al necessario aggiornamento e allenamento continuo del modello, è la sua vulnerabilità rispetto agli attacchi zero-day o, in generale, a varianti di attacchi conosciuti. Al contrario, i NIDS basati sul rilevamento di anomalie presentano numerosi falsi positivi, i quali possono generare allarmi non necessari e richiedere ulteriori verifiche, aumentando il carico di lavoro per gli operatori di sicurezza. Approcci ibridi che uniscono algoritmi basati su firme e algoritmi di anomaly detection sono stati creati per sfruttare i punti di forza di entrambe queste tecniche, mitigandone al contempo le rispettive debolezze.
In questo capitolo, nel paragrafo 3.1 si introducono modelli di apprendimento automatico in grado di automatizzare il processo di rilevamento e classificazione delle anomalie, come il **Multivariate Adaptive Regression Spline (MARS)**, le **Support Vectore Machines (SVMs)** e le **Artificial Neural Network (ANN)**. Nel paragrafo 3.2 si trattano dei dataset supervisionati composti da traffico di rete anomalo e normale, utilizzati per l'allenamento di modelli intelligenti. I dataset presenti sono il **KDD Cup 1999 Dataset** e il **NSL-KDD99 Dataset**. Successivamente nel paragrafo 3.3 viene descritto il processo di creazione, preprocessing dei dati e allenamento di un modello ensemble che unisce algoritmi MARS, SVMs e ANN [7] per la rilevazione e classificazione del traffico anomalo. Il modello utilizza le tecniche e la teoria descritta nel primo paragrafo del capitolo.

3.1 Modelli e tecniche di apprendimento automatico

3.1.1 Multivariate Adaptive Regression Splines

Il MARS (Multivariate Adaptive Regression Splines) è un modello di regressione non parametrico, introdotto da Jerome H. Friedman nel 1991, che si distingue per la sua capacità di individuare e modellare autonomamente relazioni non lineari e interazioni tra variabili. A differenza dei modelli parametrici, non richiede di specificare a priori la forma della relazione o delle interazioni, ma le scopre direttamente dai dati. È un'estensione dei modelli lineari e utilizza funzioni di base adattive, note come **hinge functions**, per costruire modelli predittivi flessibili e accurati.

MARS, dunque, crea modelli a pezzi lineari, rilevando automaticamente i punti di discontinuità, noti come **nodi** (knots). Un numero sufficiente di nodi può approssimare qualsiasi forma, ed ogni intervallo tra di essi è descritto da una funzione lineare.

Hinge function

Una *hinge function* è una funzione matematica che introduce una "discontinuità" nel modello in corrispondenza di un determinato punto (il nodo). È definita come:

$$h(x - c) = \max(0,\, x - c) \tag{3.1}$$

oppure nella forma complementare

$$h(c - x) = \max(0,\, c - x) \tag{3.2}$$

Dove x è la variabile indipendente e c è il valore del nodo, cioè il punto in cui la relazione cambia. Analizziamo come varia la funzione: quando $x > c$, la funzione (3.1) restituisce $x - c$, rappresentando una crescita lineare; mentre se $x \leq c$ restituisce 0, risultando "spenta". Analogamente, la funzione complementare in (3.2) risulta "spenta" per $x \geq c$ e produce una decrescita lineare per $x < c$.

Nei seguenti paragrafi 3.1.2 e 3.1.3 sarà analizzata la costruzione di un modello MARS.

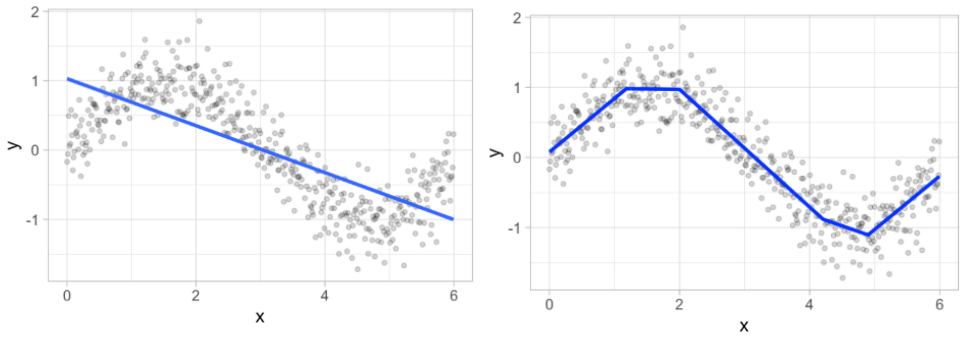
Fase di forward

Questa fase costruisce il modello aggiungendo funzioni di base in modo iterativo per ridurre l'errore residuo. Vengono eseguiti i seguenti passi:

1. Viene inizializzato un modello semplice composto solo con l'intercetta β_0 pari alla media dei valori della variabile dipendente y (Figura 3.1a):

$$f(x) = \beta_0 \quad \text{dove} \quad \beta_0 = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{3.3}$$

   Dove n rappresenta il numero di osservazioni e y_i rappresenta il valore della variabile dipendente.



(a) **(b)**

Figura 3.1. [2] La figura (a) rappresenta un esempio di modello MARS inizializzato. La figura (b) rappresenta il modello MARS alla quarta iterazione della fase di forward dove sono stati identificati i primi 4 nodi.

2. Il modello esplora le variabili indipendenti e identifica i "nodi" dove è possibile creare una funzione di base, rappresentata da una Hinge Function o dalla sua complementare. Conseguentemente, per ogni nodo c trovato e ciascuna variabile x_j vengono aggiunte:

$$B_1(x_j) = (x_j - c) \quad \text{e} \quad B_2(x_j) = (c - x_j) \tag{3.4}$$

   Dove $B(x)$ rappresenta la sopracitata funzione di base.

3. Ogni possibile combinazione di nodi, variabili e funzioni di base viene valutata per identificare quella che riduce maggiormente l'errore residuo (RSS) del modello. L'RSS (Residual Sum of Squares) rappresenta la somma dei quadrati degli errori tra i valori osservati della variabile dipendente y e i valori $\hat{y}$ predetti dal modello.

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.5}$$

4. Vengono aggiunte al modello le due funzioni di base selezionate $(B_1(x_j), B_2(x_j))$ e tramite regressione lineare vengono calcolati i coefficienti β associati.

5. Gli step (2-4) vengono ripetuti fino a quando non si raggiunge un numero massimo di funzioni di base, definito in input dell'utente, o il miglioramento del modello non è più significativo.

Alla fine della fase di forward il modello sarà definito come:

$$f(x) = \beta_0 + \sum_{k=1}^{K} \beta_k \, B_k(x) \tag{3.6}$$

Dove K è il numero massimo di funzioni di base aggiunte al modello.

Fase di backward

Alla fine della fase di forward, il modello è spesso in overfitting, poiché include tutte le funzioni che massimizzano l'adattamento ai dati. Lo scopo della fase backward è ridurre la complessità del modello eliminando le funzioni di base meno significative, mantenendo solo quelle che migliorano realmente l'accuratezza del modello. Vengono eseguiti i seguenti passi:

1. Si parte col modello completo dato in output dalla fase di forward.

2. Si elimina dal modello la funzione di base $B_i(x)$ che provoca la minima perdita di accuratezza del modello. Il metodo comune per misurare la qualità del modello è il **Generalized Cross-Validation** (GCV):

$$\text{GCV} = \frac{\text{RSS}}{\left(1 - \frac{P(M)}{N}\right)^2} \tag{3.7}$$

   Dove:

   - N è il numero di osservazioni.
   - $P(M)$ è la funzione di penalizzazione della complessità del modello che dipende dal numero di funzioni di base M. È definita come:

$$P(M) = \frac{M + (dM - 1)}{2} \tag{3.8}$$

     dove d rappresenta una penalità.

   Di conseguenza si seleziona la funzione di base da eliminare in base all'incremento minimo del valore del GCV.

3. Viene ripetuto il passo precedente eliminando una funzione alla volta fino a raggiungere un modello con un numero di funzioni di base ottimale, bilanciando la qualità del modello e la sua complessità (quindi quello che minimizza il GCV).

Alla fine della fase backward, il modello finale è più semplice e generalizza meglio sui nuovi dati.

3.1.2 Support Vector Machine

Le *Support Vector Machine* (SVM) sono un approccio per la classificazione e regressione sviluppato negli anni 90. È una generalizzazione del **classificatore a margine massimo** e si basa sull'idea di identificare un **iperpiano** che separa nel modo più netto possibile le diverse classi presenti nei dati.
Dato un insieme di esempi per l'addestramento, SVM rappresenta gli esempi come punti in uno spazio mappati in modo tale che gli esempi appartenenti alle due diverse classi siano separati il più chiaramente possibile da uno spazio, definito dall'iperpiano, che garantisce il massimo margine tra le classi. I nuovi esempi vengono quindi mappati nello stesso spazio, e la predizione della categoria alla quale appartengono viene effettuata in base al lato dell'iperpiano in cui ricadono.

Cos'è un Iperpiano?

In uno spazio n-dimensionale, un iperpiano è un sottospazio affine piatto di dimensione $n-1$. La definizione matematica di un iperpiano n-dimensionale è la seguente:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n = 0 \tag{3.9}$$

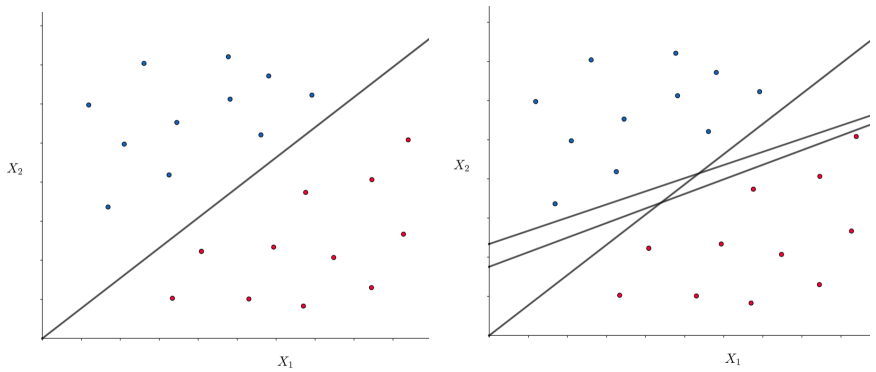
dove β_0 è il bias, $\beta_1, \beta_2, \ldots, \beta_n$ sono i coefficienti che definiscono l'orientamento dell'iperpiano nello spazio, e $X_1, X_2, \ldots, X_n$ rappresentano le variabili indipendenti. Se un punto $X = (X_1, X_2, \ldots, X_n)$ soddisfa l'equazione (3.9), allora X si troverà sull'iperpiano. Invece se

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n > 0 \tag{3.10}$$

X si troverà da una parte dell'iperpiano, mentre se

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n < 0 \tag{3.11}$$

si troverà nell'altra parte dell'iperpiano. Di fatto possiamo pensare l'iperpiano come un divisiore di uno spazio di n-dimensioni in due metà, che nel caso di una classificazione binaria rappresenteranno le due classi come mostrato nella Figura 3.2a.



(a) **(b)**

Figura 3.2. La figura (a) rappresenta un iperpiano che separa in due classi un esempio di dati. La figura (b) rappresenta lo stesso esempio di dati diviso da tre iperpiani differenti

Classificatore a Margine Massimo

Come si può osservare nella Figura 3.2, qualora un insieme di dati sia separabile da un iperpiano, esso può essere separato da un numero infinito di iperpiani. Una tecnica per selezionare quale degli infiniti iperpiani utilizzare è quella dell'**iperpiano dal massimo margine**: si cerca di individuare l'iperpiano per cui il margine, ossia la distanza dal punto più vicino di ciascuna classe, risulta il più grande possibile. Questa tecnica trova di fatto l'iperpiano in grado di generalizzare al meglio il modello, riducendo il rischio di overfitting e migliorando la separabilità tra le classi.

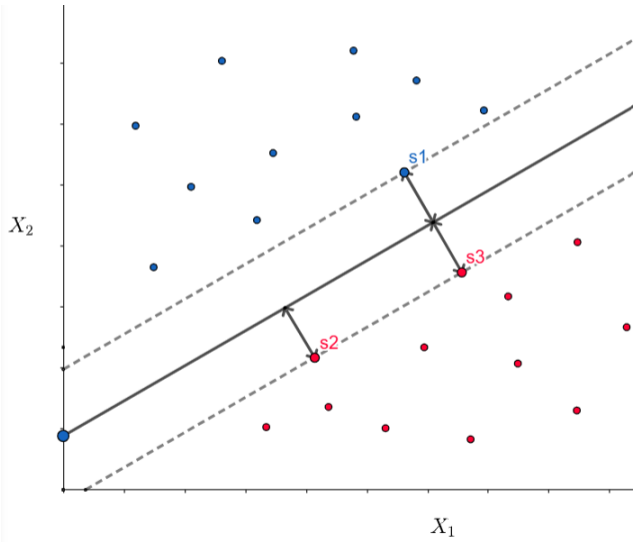


Figura 3.3. Date le due classi di osservazione, la linea solida rappresenta l'iperpiano massimale. Il margine è rappresentato dalla distanza dall'iperpiano alle rette tratteggiate. I punti $s1, s2, s3$ situati sulla linea tratteggiata hanno il ruolo di vettori di supporto.

Possiamo poi classificare un set di dati in base alla loro posizione rispetto all'iperpiano massimale; questo approccio è noto come *classificatore a margine massimo*. Le osservazioni che definiscono la distanza marginale massima sono denominati **vettori di supporto**. Questi ultimi, di fatto, "supportano" l'iperpiano massimale ed un loro minimo spostamento ne modificherebbe la posizione, influenzando direttamente la classificazione dei nuovi dati.

Support Vector Classifier

Non sempre i dati appartenenti a due classi sono separabili da un iperpiano. Questo può accadere quando un dato di una delle due classi si trova all'interno della regione occupata dall'altra classe, rendendo impossibile una separazione lineare netta. Di conseguenza il Classificatore a Margine Massimo è estremamente sensibile al variare di un singolo dato che potrebbe portare il modello ad overfittare o direttamente rendere impossibile la classificazione. Il *Support Vector Classifier* (o classificatore soft-margin) considera un iperpiano che, seppur non separando le classi perfettamente, è più robusto rispetto ai singoli dati. Il modello permette ad alcune osservazioni di trovarsi non solo nella regione del margine, ma anche nella parte errata dell'iperpiano.

L'iperpiano scelto separa con successo la maggior parte dei dati nelle due classi, ma potrebbe classificare erroneamente qualche osservazione. Segue la costruzione di un SVC: dato un insieme di n dati di allenamento $x_1, \ldots, x_n \in \mathbb{R}^p$ associati ai rispettivi label $y_1, \ldots, y_n \in \{-1, 1\}$ l'iperpiano è la soluzione del seguente problema di ottimizzazione:

$$\underset{\beta_0,\beta_1,\ldots,\beta_n,\xi_1,\ldots,\xi_n,M}{\text{massimizza}} \quad M \tag{3.12}$$

Soggetto ai vincoli:

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{3.13}$$

$$y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) \geq M(1 - \xi_i), \quad \forall i = 1, \ldots, n \tag{3.14}$$

$$\xi_i \geq 0, \quad \sum_{i=1}^{n} \xi_i \leq C \tag{3.15}$$

Dove in (3.12) M è la larghezza del margine massimo. Il vincolo in (3.13) serve a normalizzare il vettore dei coefficienti β, garantendo che la soluzione trovata sia scalata in modo appropriato. Una variabile *slack* ξ_i definisce dove l'i-esimo dato è situato rispetto all'iperpiano ed al margine:

$$\begin{cases} \xi_i = 0, & x_i \text{ è correttamente classificata e al di fuori del margine} \\ 0 < \xi_i \leq 1, & x_i \text{ è all'interno del margine e classificato correttamente} \\ \xi_i > 1, & x_i \text{ è nella regione sbagliata dell'iperpiano} \end{cases} \tag{3.16}$$

In (3.15) C svolge il ruolo di *parametro di regolarizzazione* e rappresenta il compromesso tra massimizzazione del margine e minimizzazione degli errori di classificazione. Per $C > 0$ non piu di C osservazioni possono essere dal lato sbagliato dell'iperpiano. Più il valore è alto più largo sarà il margine e la tolleranza del modello, più il valore è basso maggiore è il rischio di overfit del modello. Una volta risolto (3.12)-(3.15), classifichiamo un osservazione x basandoci sul segno ottenuto calcolando $f(x) = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$ (in poche parole in base alla regione dove è situato rispetto all'iperpiano). Questo prrocesso si basa sul **prodotto scalare** tra il vettore dell'osservazione x e il vettore dei coefficienti $w = (\beta_1, ..., \beta_n)$ che definisce l'iperpiano. Il prodotto scalare di due osservazioni $x_i, x_{i'}$ è dato da:

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \tag{3.17}$$

L'SVC può essere rappresentato come:

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \langle x, x_i \rangle \tag{3.18}$$

Dove:

- α_i sono i moltiplicatori di Lagrange ottenuti da (3.12)-(3.15). Determinano il peso con cui ogni vettore x_i contribuisce alla decisione, solo i vettori di supporto (quelli con $\alpha_i \neq 0$) hanno un impatto sulla funzione.
- $\langle x, x_i \rangle$ è il prodotto scalare dell osservazione x e uno dei vettori x_i.

Support Vector Machine con kernel

Il Support Vector Classifier è efficace quando i dati sono linearmente separabili (o quasi separabili con il soft-margin). Tuttavia nella maggior parte dei casi quando i dati sono di alte dimensioni, o in genere nella rilevazione e classificazione di traffico di rete, si ha a che fare con osservazioni non separabili in modo lineare. Il *Support Vector Machine (SVM)* è un estensione del SVC che permette di gestire dati non linearmente separabili grazie all'uso delle **funzioni di kernel**. Queste funzioni permettono di proiettare i dati in uno spazio a dimensionalità superiore, in cui la separabilità lineare diventa possibile, senza calcolare esplicitamente la trasformazione. Le SVM introducono il *kernel trick* che permette di sostituire il prodotto scalare $\langle x_i, x_j \rangle$ in (3.18) con una funzione di kernel $K(x_i, x_j)$:

$$K(x_i, x_{i'}) = \phi(x_i)\phi(x_j) \tag{3.19}$$

Dove $\phi(x)$ rappresenta la trasformazione implicita in uno spazio di dimensione superiore.

Le funzioni di kernel più utilizzate per separare dati con relazioni non lineari sono:

- **Kernel Radiale (RBF):**

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right) \tag{3.20}$$

  Dove γ è una costante.

- **Kernel Polinomiale:**

$$K(x_i, x_j) = (x_i \cdot x_j + c)^d \tag{3.21}$$

  Dove d è il grado del polinomio e c è un termine di regolazione che determina il contributo della componente lineare nel prodotto scalare.

3.1.3 Artificial Neural Network

Una *Rete Neurale Artificiale (ANN)* è un modello computazionale ispirato alla struttura del cervello umano. Così come la sua controparte biologica, una ANN è composta da neuroni artificiali organizzati in strati (*layers*) che elaborano e trasformano l'informazione attraverso connessioni ponderate. La struttura di una ANN è raffigurata nella Figura 3.4.

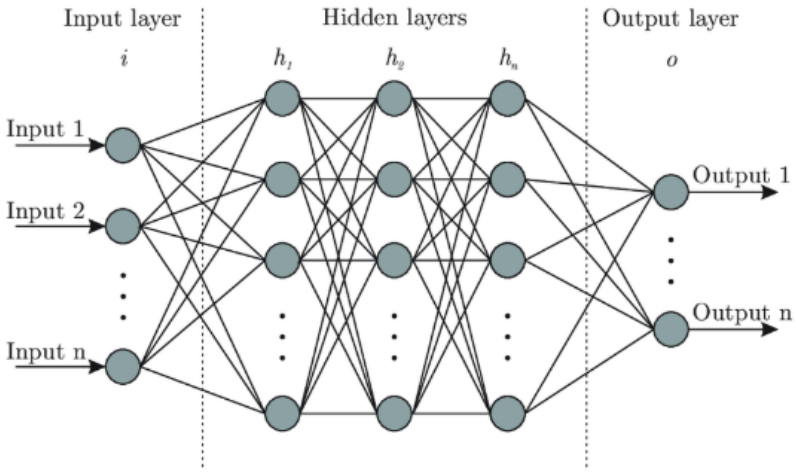


Figura 3.4. Architettura di un Artificial Neural Network

L'insieme dei segnali che la ANN riceve in input si propaga attraverso gli strati intermedi (*hidden layers*) venendo trasformato progressivamente fino a raggiungere lo strato di Output.

Neuroni artificiali

Le reti neurali sono costruite tramite connessione di **neuroni artificiali** che, ricevendo in input uno stimolo, lo trasformano attraverso operazioni matematiche e lo propagano agli strati successivi della rete. Un neurone artificiale è strutturato come mostrato dalla Figura 3.5.

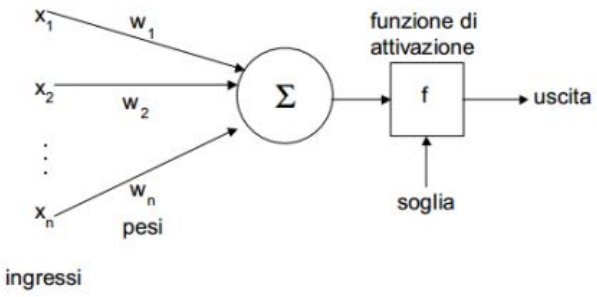


Figura 3.5. Struttura di un neurone artificiale

Dove:

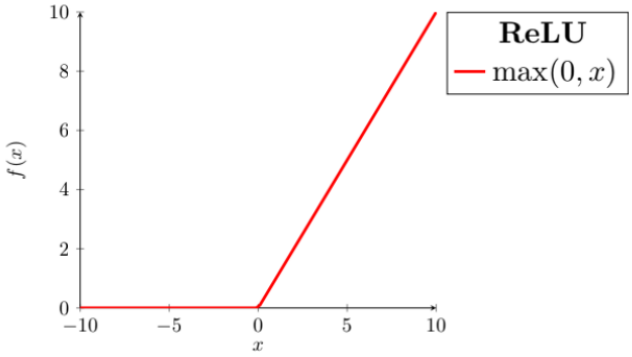
- $x_1, ..., x_n$ sono i valori ricevuti in input dal neurone.
- $w_1, ..., w_n$ sono i pesi applicati ai valori di input.
- f è la funzione di attivazione che determina come e quanto il segnale elaborato dal neurone viene propagato agli strati successivi della rete.

Dunque un neurone è definito matematicamente come:

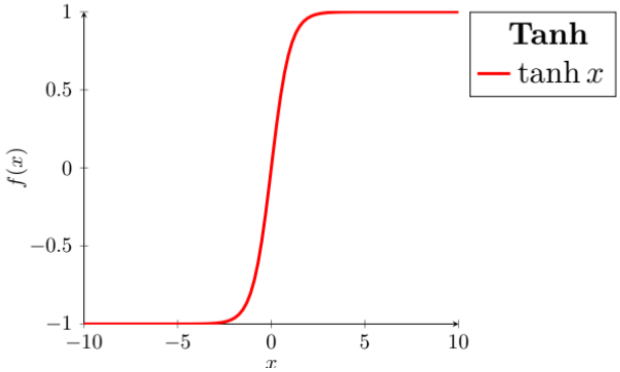
$$z = \sum_{i=1}^{n} w_i x_i + b, \qquad y = f(z) \tag{3.22}$$

Con b che rappresenta il bias (la soglia). A seconda della funzione di attivazione utilizzata, il comportamento della rete cambia. Alcune delle funzioni più comuni sono:

- **Funzione Softmax:** utilizzata dai neuroni in output per la classificazione "multi-classe"
$$f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{3.23}$$
- **Funzione ReLU:** famosa per la sua efficienza computazionale.



- **Funzione Tanh**



Allenare una rete neurale

L'addestramento di una rete neurale supervisionata consiste nell'ottimizzazione dei pesi in funzione dei dati di addestramento, affinchè il modello impari a generalizzare le relazioni presenti nei dati, minimizzando l'errore tra le previsioni e i valori reali. Questo processo avviene attraverso due fasi ripetute per un tot di epoche:

1. **Forward Propagation:** In questa fase, i dati di input vengono passati attraverso la rete neurale, strato dopo strato, fino a generare un output. Durante la propagazione, ogni neurone applica la propria funzione di attivazione, combinando i segnali in ingresso pesati, fino ad ottenere una stima finale. Il risultato viene confrontato con il valore reale, calcolando così l'errore della previsione.

2. **Backpropagation:** Una volta calcolato l'errore, questo viene propagato all'indietro nella rete per aggiornare i pesi. Attraverso algoritmi di ottimizzazione i pesi vengono modificati in modo da ridurre l'errore nelle previsioni future.

La tecnica per calcolare l'errore della previsione varia in base al problema di classificazione che si cerca di risolvere, la più diffusa nella multi-classficazione è la *Cross-Entropy Loss*:

$$H(\hat{y}, y) = -\sum_{i=1}^{K} \hat{y}_i \log(y_i) \tag{3.24}$$

Dove K è il numero delle classi, $\hat{y}$ è il vettore one-hot che rappresenta la classe reale ed y_i è la probabilità prevista per la classe i. La misura dell'errore derivata da una funzione di perdita svolge anche il ruolo di guida durante il processo di ottimizzazione, fornendo un feedback al modello sulla qualità dell'adattamento ai dati. Minore è l'errore, migliore sarà la capacità predittiva del modello. Il risultato della loss function è utilizzato dall'ottimizzatore, un algoritmo con il ruolo di massimizzare l'efficienza del modello, per aggiornare i pesi della rete.

Stochastic Gradient Descent (SGD)

Lo Stochastic Gradient Descent è un ottimizzatore che lavora calcolando il gradiente della loss function rispetto ai pesi e ai bias, utilizzando un sottoinsieme dei dati (mini-batch) per ogni aggiornamento. Questo permette di rendere il processo di aggiornamento più efficiente evitando di rimanere bloccati in minimi locali. Il gradiente indica la direzione in cui la funzione di loss aumenta, l'SGD modifica i pesi nella direzione opposta ad esso, riducendo progressivamente l'errore e migliorando l'efficienza del modello.

Segue lo pseudocodice dell'SGD:

Algorithm 1 Stochastic Gradient Descent (SGD)

Input: Dati di training $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, funzione di loss $Lf(\theta)$, tasso di apprendimento η, numero di epoche E
Inizializza i parametri del modello θ in modo casuale
for epoca = 1 **to** E **do**
    Mescola l'insieme di training
    for ogni esempio $(x^{(i)}, y^{(i)})$ nell'insieme di training **do**
        Calcola il gradiente: $g \leftarrow \nabla_\theta Lf(f(x^{(i)}; \theta), y^{(i)})$
        Aggiorna i parametri: $\theta \leftarrow \theta - \eta \cdot g$
    end for
end for
Output: Parametri θ ottimizzati

Prevenire l'overfit: Regolarizzazione

Uno degli ostacoli più comuni dell'allenamento ed efficienza di una rete neurale è l'**overfitting**. Questo fenomeno accade quando la rete si adatta troppo strettamente ai dati di allenamento e non impara a generalizzare su dati nuovi. Per evitare l'overfitting, vengono impiegate diverse tecniche di regolarizzazione come il *Dropout* e l'*early stopping*.

Dropout: tecnica di regolarizzazione che disattiva una certa percentuale di neuroni durante ogni epoca di training; questo riduce la dipendenza della rete da neuroni o features specifiche, rendendola più robusta e meno suscettibile all'overfitting.

Early stopping: tecnica che monitora la funzione di perdita sui dati di validazione durante il training e interrompe l'addestramento quando l'errore inizia ad aumentare. Questo evita che la rete neurale continui ad apprendere pattern specifici dei dati di allenamento, migliorando la capacità di generalizzazione del modello.

3.1.4 Ensemble learning

L'*Ensemble learning* è una tecnica di machine learning che aggrega due o più modelli. In poche parole, combina più modelli per cercare di migliorare le previsioni dei singoli.

L'ensemble learning si divide in due categorie:

- **Ensemble paralleli**: allenano i modelli in parallelo e in modo indipendente l'uno dall'altro.
- **Ensemble sequenziali**: addestrano modelli in serie, dove ogni modello successivo cerca di correggere gli errori del precedente.

Ma come avviene la decisione sull'output dei modelli combinati? Una delle tecniche più comuni è quella del **voto per maggioranza**. Ogni modello base effettua una predizione, l'output sarà determinato in base alla predizione più frequente tra tutti i modelli. Una variante di questo approccio è il **voto per maggioranza pesato** che assegna un peso differente a ciascun modello in base alla sua affidabilità o accuratezza. In questo modo, le predizioni dei modelli più performanti avranno un impatto maggiore sulla decisione finale, che viene determinata dalla somma dei pesi associati alle varie classi, scegliendo quella con il peso complessivo più elevato.

3.2 Datasets

3.2.1 KDD Cup 1999

Il *KDD99 Data* [1] è uno dei dataset più utilizzato per il training e ricerche di algoritmi di apprendimento automatico per Network Intrusion Detection Systems. Creato per la competizione *KDD Cup 1999* è basato sul traffico di rete ottenuto dal *DARPA 1998 Intrusion Detection Evaluation Program* [5] simulando una tipica LAN dell'aeronautica americana. Il flusso di dati è stato catturato per un periodo di sette settimane, includendo sia traffico di rete legittimo che simulazioni di attacchi. Questi ultimi sono stati generati con l'obiettivo di rappresentare scenari reali di intrusione, rendendo il dataset ottimo per l'allenamento di NIDS.

Struttura del dataset

La struttura del dataset è raffigurata della Tabella 3.6 ([6, p. 691]), presenta 41 features più label e viene suddivisa in quattro parti. Le features di base rappresentano informazioni generali sulla connessione come la durata della connessione, il protocollo utilizzato ecc. . . La seconda "categoria" di features contiene statistiche utili per rilevare varie tipologie d'attacco al sistema come il numero di tentativi di login falliti o se quest'ultimo ha avuto successo, il numero di shell aperte o se l'accesso è avvenuto come root. Solitamente attacchi di tipo DoS e scanning generano un numero molto elevato di connessioni in un periodo breve. Le due categorie successive hanno una finestra temporale di due secondi utili per catturare questi tipi di attacco che agiscono nel breve termine.

Il training dataset di $4,898,431$ records presenta labels di tipo normale e 22 tipi diversi di attacchi. Il traffico può essere diviso in cinque categorie diverse:

Categoria	**Numero di istanze**	**Percentuale (%)**
Normale	972,781	19.86
DoS	3,883,370	79.26
Probing	41,102	0.84
R2L	1,126	0.02
U2R	52	0.001
Totale	4,898,431	100

Tabella 3.1. Distribuzione delle istanze nel dataset KDD99 per categoria di attacco.

Di seguito l'elenco e descrizione dei vari tipi di attacco per ogni categoria [6, p. 690-696].

Denial-Of-Service

Un attacco DoS impedisce o compromette l'uso autorizzato di reti, sistemi o applicazioni esaurendo risorse come unità di elaborazione centrale (CPU), memoria, larghezza di banda e spazio su disco.

Category Name	Features Names	Type	Description
Basic Features	P1. Duration	Integer	Duration of the connection (seconds)
	P2. Protocol_type	Nominal	Type of Protocol (TCP, UDP, ICMP etc.)
	P3. Service	Nominal	Network Service (http, telnet, http, others)
	P4. Flag*	Nominal	Connection status (SF, S0, S1, S2, S3, OTH, REJ, RSTO, RSTO,S0, SH, RSTRH, SHR)
	P5. Src_bytes	Integer	Number of bytes sent from source to destination
	P6. Dst_bytes	Integer	Number of bytes received from destination
	P7. Land	Binary	If source and destination IP are identical. It is 1 else 0
	P8. Wrong_fragment	Integer	Sum of Bad checksum packets in a connection
	P9. Urgent	Integer	Some of packets where urgent bit is set 1.
Content Feature	P10. Hot	Integer	Sum of hot actions in a connection (entering a system directory, creating programs and executing programs)
	P11. Num Failed Login	Integer	Number of failed logins in a connection
	P12. Logged in	Binary	1 if successful login else 0.
	P13 Num compromised	Integer	Sum of not found error appearances in a connection
	P14. Root shell	Binary	1 if root shell is obtained else 0
	P15. Su attempted	Binary	Its 1 if su command attempted else 0
	P16. Num root	Integer	Sum of operations performs as a root in a connection
	P17. Num file creation	Integer	Sum of file creations in a connection
	P18. Num shells	Integer	Number of shell prompts
	P19.Num access files	Integer	Sum of operations in control files in a connection
	P20.Num outbound cmds	Integer	Sum of outbound commands in a ftp session
	P21.Is hot login	Binary	If the user is accessing s root , it is set to 1 else 0
	P22.Is guest login	Binary	If the user is accessing s guest, it is set to 1 else 0
Traffic Feature (2s time window)	P23. Count	Integer	Sum of connections to the same destination IP
	P24. Srv count	Integer	Sum of connections to the same destination Port no.
	P25. Serror rate	Real	Percentage of connections that have activated the flag (P4) s0, s1, s2 or s3, among the connections aggregated in count (P23)
	P26. Srv serror rate	Real	Percentage of connections that have activated the flag (P4) s0, s1, s2 or s3, among the connections aggregated in srv_count (P24)
	P27. Rerror rate	Real	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in count (P23)
	P28. Srv rerror rate	Real	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in count (P23)
	P29. Same srv rate	Real	Percentage of connections that were to the same service, among the connections aggregated in count (P23)
	P30. Diff srv rate	Real	Percentage of connections that were to different services, among the connections aggregated in count (P23)
	P31. Srv diff host rate	Real	Percentage of connections that were to different destination machines among the connections aggregated in srv_count (P24)
Traffic Feature (2s time window from dest. to host)	P32. Dst host count	Integer	Sum of connections to the same destination IP address
	P33. Dst host srv count	Integer	Sum of connections to the same destination port number
	P34. Dst host same srv rate	Real	Percentage of connections that were to the same service, among the connections aggregated in dst_host_count (P32)
	P35. Dst host diff srv rate	Real	Percentage of connections that were to different services, among the connections aggregated in dst_host_count (P32)
	P36. Dst host same src port rate	Real	Percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (P33)
	P37. Dst host srv diff host rate	Real	Percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (P33)
	P38. Dst host serror rate	Real	Percentage of connections that have activated the flag (f4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (P32)
	P39. Dst host srv serror rate	Real	Percent of connections that have activated the flag (P4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (P33)
	P40. Dst host rerror rate	Real	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in dst_host_count (P32)
	P41. Dst host srv rerror rate	Real	Percentage of connections that have activated the flag (P4) REJ, among the connections aggregated in dst_host_srv_count (P33)

Figura 3.6. TCP Connection Features in KDD99

Gli attacchi Denial-of-Service presenti nel dataset sono:

- **Smurf:** un attacco di amplificazione dove un attaccante invia ad un IP broadcast un grande numero di messaggio ICMP echo, utilizzando come l'IP della vittima come indirizzo sorgente.

- **Land:** l'attaccante invia un pacchetto SYN l'ip sorgente uguale a quello di destinazione.

- **Teardrop:** l'attaccante invia una serie di pacchetti frammentati alla macchina bersaglio. L'offset dei frammenti è impostato per far si che i pacchetti si sovrappongano tra di loro, in alcune sistemi questo porta ad un crash.
- **Ping of Death:** viene inviato un pacchetto piu pesante del limite massimo, questo causa il crash del sistema.
- **Mailbomb:** in questo attacco un utente non autorizzato invia un enorme numero di email con allegati molto pesanti ad un determinato mail server. Questo causa il disco del server pieno con conseguente servizio mail bloccato.
- **SYN flood:** in questo attacco viene sfruttata una vulnerabilità del protocollo TCP. L'attaccante invia una richiesta SYN alla vittima che di conseguenza invia un ACK ed aspetta. Il server nella tabella TCP inserisce i dati della connessione pendente che invece non risponde. Questo causa la tabella a finire lo spazio e non permettere nuove connessioni.
- **Neptune:** non è un attacco DoS diretto, ma un malware che può essere usato per creare una botnet. Una volta infettati, questi sistemi possono vengono comandati per eseguire un attacco DDoS.

Probe

Un attacco di tipo scan è la prima fase dove l'attaccante studia la vittima per trovare vulnerabilità accessibili cosi da effettuare la vera e propria intrusione. Gli attacchi scanning presenti sono:

- **Ipsweep:** utilizzato per individuare quali host si trovano in ascolto nella rete, effettuato inviando vari pacchetti ping. Se l'host risponde, la risposta rivela il suo ip.
- **Reset Scan:** l'attaccante invia pacchetti reset (RST flag) alla vittima per scoprire se è attiva.
- **SYN scan:** l'attaccante invia un grande numero di pacchetti SYN a varie porte. Le porte aperte rispondono con un ACK mentre quelle chiuse con RST.
- **Satan:** SATAN è uno strumento di analisi della sicurezza. Gli attaccanti lo sfruttano per inviare richieste a porte e servizi così da identificarne vulnerabilità e configurazioni errate.

User To Root (U2R)

Ugli attacchi U2R sfruttano vulnerabilità per ottenere i privilegi root sul sistema.

- **Buffer Overflow:** l'attaccante sfrutta codice/file vulnerabili per far eseguire codice assembly, quest'ultimo spinge il sistema ad aprire una shell con i privilegi del file.
- **Rootkit:** sono kit, ovvero strumenti o insiemi di strumenti, come sequenze di macro o veri e propri software, atti ad ottenere sul computer bersaglio i permessi di root.

- **Loadmodule:** tecnica in cui un aggressore sfrutta le funzionalità di caricamento dinamico dei moduli per iniettare ed eseguire codice malevolo all'interno di un processo legittimo.
- **Perl:** se i nomi o i percorsi dei moduli (.pm) non sono controllati correttamente, un aggressore potrebbe forzare il caricamento di codice malevolo.

Remote to Local (R2L)

Neli attacchi R2L, un aggressore remoto sfrutta vulnerabilità o debolezze di sicurezza per ottenere accesso locale ad un sistema target. Una volta compromesso, l'attaccante può eseguire comandi, installare software malevolo o compromettere ulteriormente il sistema.

- **Warezmaster:** l'attaccante sfrutta un bug presente nei server FTP, carica file malevoli noti come "warez" che poi verranno scaricati dagli utenti.
- **PHF:** exploit che sfrutta vulnerabilità in alcuni script CGI. L'attaccante invia richieste appositamente costruite a questi script vulnerabili per ottenere l'accesso al codice sorgente o eseguire comandi arbitrari.
- **FTP_write:** l'attaccante sfrutta una vulnerabilità o configurazione errata in un server FTP per scrivere file arbitrari sul server. Sfruttato anche per creare backdoors.

Problematiche del KDD99

Nonostante sia il dataset più utilizzato nel panorama degli Intrusion Detection Systems, il KDD99 presenta numerose criticità. Dalle più immediate come l'elevato sbilanciamento delle classi (Tabella 3.1) e l'inevitabile elevato tempo di training dato dal numero di dati. Il problema fondamentale del dataset è l'enorme numero di duplicati [3]. In particolare, il set di addestramento risulta costituito per il 78% da istanze duplicate, con la classe DoS caratterizzata da un tasso di ridondanza pari all'86%. La presenza di dati duplicati distorce l'addestramento dei modelli, portando a un'overfitting sui pattern più comuni e limitando la capacità del modello di generalizzare su nuovi attacchi.

3.2.2 NSL-KDD99 Dataset

Nel 2009 viene creato il *NSL-KDD* [8] per risolvere le criticità del KDD99. Questo nuovo dataset elimina le ridondanze del suo predecessore e contiene un numero più moderato di dati, rendendolo così più leggero e più adatto alla sperimentazione. La distribuzione delle classi nel training set, tuttavia, rimane sbilanciata (Tabella 3.3).

Categoria	**Numero di istanze**	**Percentuale (%)**
Normale	67,343	53.46
DoS	45,927	36.49
Probing	11,656	9.26
R2L	995	0.79
U2R	52	0.04
Totale	125,973	100

Tabella 3.2. Distribuzione delle istanze nel training set del NSL-KDD.

Categoria	**Numero di istanze**	**Percentuale (%)**
Normale	9,710	43.08
DoS	7,458	33.09
Probing	2,421	10.74
R2L	2,754	12.22
U2R	200	0.89
Totale	22,543	100

Tabella 3.3. Distribuzione delle istanze nel test set NSL-KDD.

L'NSL-KDD mantiene la struttura originale del KDD99, mantenendo le 41 features (Tabella 3.6). I label delle classi rimangono invariati, ma le istanze sono già suddivise nelle cinque categorie: Probe, DoS, Normal, R2L e U2R.

3.3 Ensemble di modelli di Machine Learning

Nel seguente paragrafo descrivo il processo di creazione di un modello basato sull'ensemble parallelo di algoritmi MARS, SVM e ANN. Il suddetto modello avrà il ruolo di analizzatore in un Network Intrusion Detection System ibrido, quindi avrà il compito di determinare se il traffico osservato è anomalo o normale. Essendo ibrido unisce i vantaggi dell'approcco signature-based e anomaly-based, determinando se è in corso un attacco per poi categorizzarlo in tre categorie: DoS, Probe e R2L. Per ogni componente del modello viene descritta la motivazione dell'uso e l'architettura completa.
Al fine di descrivere la selezione di quest'ulltima, introduco le metriche chiave nella valutazione di un modello:

- **Accuracy**: misura la percentuale di istanze correttamente classificate rispetto al totale delle istanze analizzate.

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

  Dove TP sono i veri positivi, TN i veri negativi, FP i falsi positivi e FN i falsi negativi.

- **Recall (Tasso di Rilevamento)**: rappresenta la capacità del modello di individuare correttamente le istanze positive.

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

  Un alto valore di recall implica che il modello rileva la maggior parte delle intrusioni, minimizzando i falsi negativi.

- **F1-Score**: è la media armonica tra precisione e recall ed è utilizzata per bilanciare i due indicatori nei dataset sbilanciati.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

  Un elevato F1-score indica che il modello mantiene un buon equilibrio tra la capacità di rilevare attacchi e la riduzione di falsi positivi.

3.3.1 Pre-processing e gestione del dataset

Prima di descrivere, allenare e testare i vari componenti dell'ensemble, è necessario che i dati siano preparati correttamente per garantire l'efficienza e la coerenza dei modelli. La **normalizzazione** viene utilizzata per ridurre la scala dei dati in modo che siano comparabili tra loro. La maggior parte dei modelli è molto sensibile alla normalizzazione, ad esempio le ANN richiedono che il dato sia normalizzato o standardizzato per performare bene. Per questo motivo, ho utilizzato la normalizzazione tramite Z-score, definita come:

$$Z = \frac{X - \mu}{\sigma}$$

dove:

- X è il valore originale della feature,
- μ è la media della feature,
- σ è la sua deviazione standard.

Un altro fattore che influenza direttamente l'efficienza dei modelli è la distribuzione delle classi nel dataset. Questo fenomeno può portare il modello a privilegiare la classe più frequente, riducendo la sua sensibilità nella classificazione delle categorie minoritarie. Per l'allenamento e il testing delle componenti dell'ensemble, ho utilizzato rispettivamente il NSL-KDD Training Set e il NSL-KDD Test Set. Come si evince dalla Tabella 3.3, il dataset risulta sbilanciato. Questo porterebbe diversi problemi soprattutto nei modelli binari. Per risolvere questo problema ho suddiviso il dataset in quattro "campioni" di training bilanciati, rispettivamente uno per ogni classe, tenendo conto delle varie relazioni tra i dati e il label. Ad esempio gli attacchi R2L risultano simili al traffico normale [6], di conseguenza il dataset di allenamento per i modelli binari R2L sarà distribuito come segue: 50% R2L, 40% normal, 5% Probe e 5% DoS. Una classe estremamente sottodimesionata risulta quella U2R che rappresenta solamente il 0,004% del training set e il 0,89% del test set, Questa distribuzione estremamente sbilanciata renderebbe difficile, se non impossibile, allenare un modello efficace, poiché il numero di dati non permetterebbe di apprendere pattern significativi. Per questo motivo ho deciso di escludere gli attacchi U2R dalla classificazione dell'ensemble. Tuttavia i dati relativi a quasta classe non saranno sprecati, li utilizzerò per testare come l'ensemble reagisce ad attacchi non presenti nell'allenamento (simulazione attacchi "zero-day").

3.3.2 Componente SVM

Il primo componente dell'ensemble è formato da quattro modelli binari SVM: Normal/Anomalo, DoS/Rest, Probe/Rest e R2L/Rest. I modelli SVM sono particolarmente efficaci nel panorama dei *Network Intrusion Detection Systems*, permettendo di modellare relazioni non lineari ed ad alta dimensionalità. Grazie al modello Normal/Anomalo, il cui compito è distinguere tra traffico lecito e potenzialmente malevolo, il componente acquisisce robustezza nei confronti degli attacchi *zero-day*. Una volta individuata un anomalia, i tre modelli successivi consentono di classificare l'attacco, combinando in questo modo gli approcci signature-based e anomaly-based. L'approccio di suddividere il sistema in quattro modelli binari, piuttosto che adottare un'unica SVM multi-classe, non si limita alla notevole efficacia nella separazione delle classi, ma garantisce anche una maggiore scalabilità del sistema. Di fatto una volta implementato il componente, esso potrà aggiungere modelli per classificare nuovi tipi di attacco senza dover necessariamente riallenare il componente intero. I modelli sono stati implementati in Python utilizzando la libreria *scikit-learn*.

Scelta dell'architettura dei modelli binari

La ricerca di un'architettura efficace di un SVM rappresenta un aspetto fondamentale. La scelta della funzione di kernel è cruciale per riuscire a separare le classi nel modo piu efficiente possibile. Per la selezione dell'architettura, ho analizzato il

comportamento di ciascun modello al variare della funzione di kernel, confrontando le metriche di valutazione ottenute sul test set. La statistica chiave per la valutazione delle prestazioni del modello è il recall, soprattuto nel modello Anomalo/Normale dove è fondamentale minimizzare il numero di attacchi non rilevati. Le funzioni di kernel testate per ogni modello sono: *RBF*, polinomiale di grado 2 e 3.

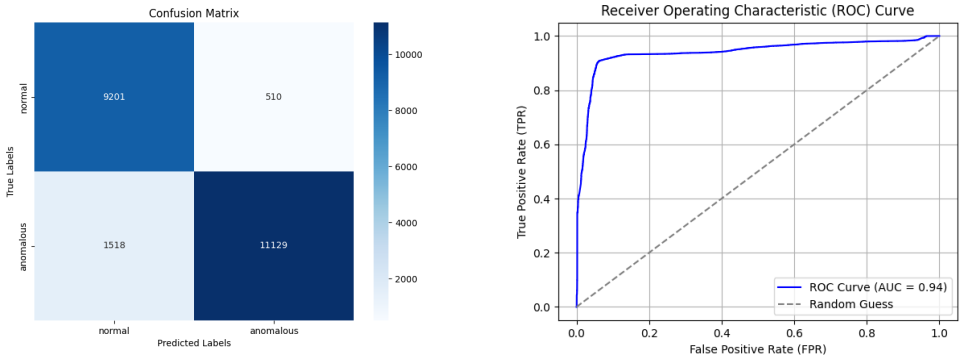


Figura 3.7. Classificatore Anomalo/Normale

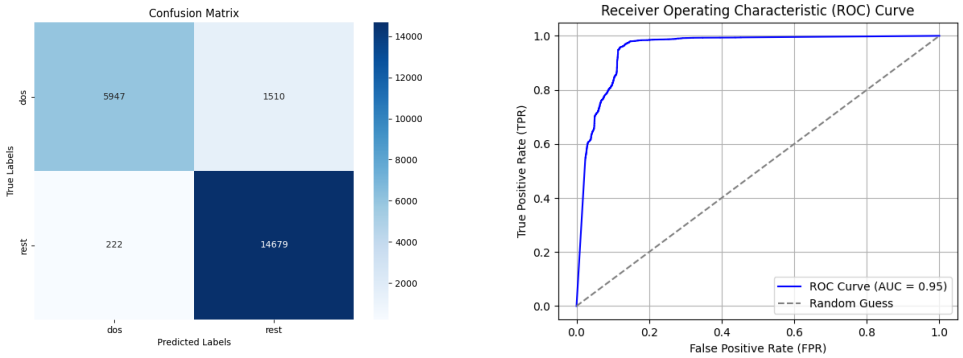


Figura 3.8. Classificatore DoS/Rest

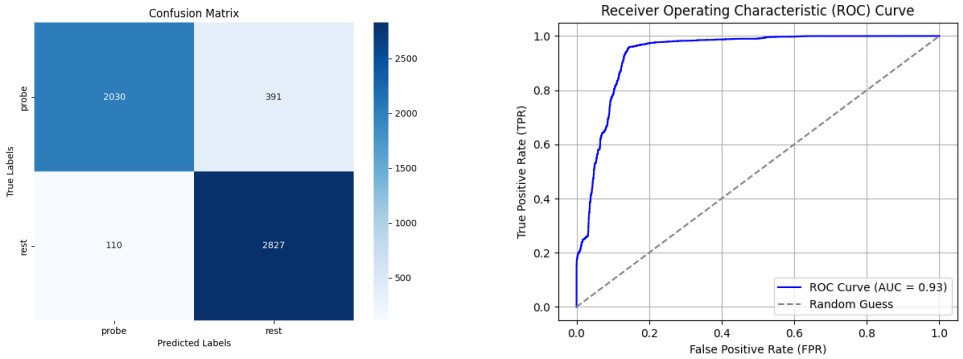


Figura 3.9. Classificatore Probe/Rest

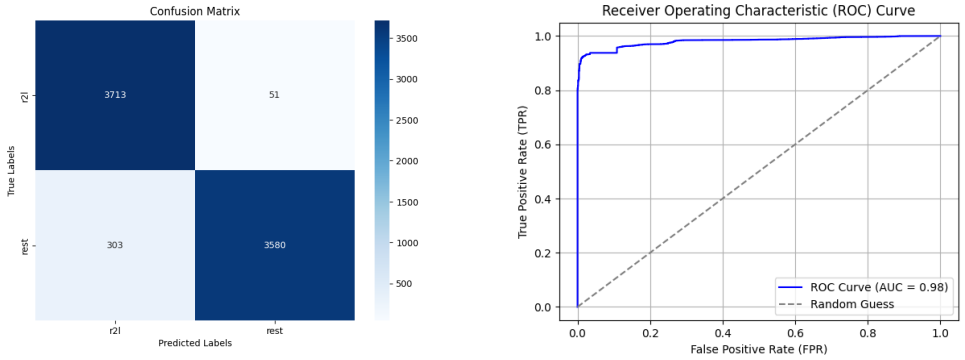


Figura 3.10. Classificatore R2L/Rest

La Tabella 3.5 contiene i valori delle metriche dei vari modelli che andranno a comporre il componente SVM. Come rappresentato i modelli non utilizzano la stessa funzione di kernel evidenziando un altro vantaggio dato dalla divisione in modelli binari: la possibilità di ottimizzare indipendentemente ogni classificatore, selezionando la funzione di kernel più adatta per ciascuna sottocategoria di attacco.

Modello	**Kernel**	**Accuracy (%)**	**Recall (%)**	**F1-Score (%)**
Normal/Anomalo	POLY3	92.0	91.0	91.0
DoS/Rest	POLY3	92.0	92.0	92.0
Probe/Rest	POLY3	92.0	92.0	92.0
R2L/Rest	RBF	96.0	95.0	95.0

Tabella 3.4. Metriche di valutazione per i quattro modelli SVM.

3.3.3 Componente MARS

Il secondo componente dell'ensemble, come il primo, è formato da quattro modelli binari MARS: Normal/Anomalo, DoS/Rest, Probe/Rest e R2L/Rest. I modelli MARS, grazie alla lorò capacità di definire funzioni a pezzi, sono in grado di modellare relazioni non lineari ad alta dimensionalità. Presentano gli stessi vantaggi del componente SVM combinando gli approcci signature-based e anomaly-based. I modelli sono stati implementati in R utilizzando la libreria *earth*.

Scelta dell'architettura dei modelli binari

Grazie alla fase di forward, il modello MARS è in grado di adattarsi automaticamente alla complessità dei dati, identificando sia il numero ottimale di nodi sia la loro posizione ideale. Durante la scelta dell'architettura, di conseguenza, inizializziamo il modello con un grande numero di nodi. Durante la fase di backward il modello selezionerà solo i nodi fondamentali bilanciando l'efficienza del modello e la sua capacità di generalizzazione. Tuttavia è necessario definire alcuni parametri per

ottimizzare: i *gradi* del modello aggiungono una "curvatura" attraverso funzioni di ordine superiore permettendo di rappresentare relazioni più articolate (*degrees*); il numero massimo di funzioni base incluse nel modello dopo la fase di backward (*nprune)*. Per identificare le combinazioni migliori per ogni modello, ho utilizzato la tecnica del **Grid-Search**: si definisce un insieme di valori possibili per ciascun iperparametro e si eseguono tutte le combinazioni possibili, valutando le prestazioni del modello per ciascuna configurazione. Come per il componente precedente, la scelta ricade sulle configurazioni con il *recall* più alto.

Modello	Degree	nprune	Accuracy (%)	Recall (%)	F1-Score (%)	AUC
N/A	3	18	93.0	96.0	93.0	96
DoS/R	2	50	88.0	94.0	85.0	95
Probe/R	1	15	94.0	90.0	80.0	93
R2L/R	3	15	92.0	92.0	92.0	95

Tabella 3.5. Metriche di valutazione per i quattro modelli MARS.

3.3.4 Componente ANN

L'ultimo componente è una rete neurale. Le ANN sono altamente utilizzate per approcci signature-based grazie alla loro capacità di apprendere pattern complessi e identificare caratteristiche distintive nei dati. Questo modello, dunque, rafforza l'approccio basato su firme dell'ensemble. Tuttavia, le ANN soffrono particolarmente lo sbilanciamento dei dati, poiché tendono ad adattarsi alla classe predominante, penalizzando la capacità di riconoscere le classi meno rappresentate. La Tabella 3.6 raffigura l'architettura della rete neurale composta da: 41 neuroni in input date dal numero di features, 59 neuroni nell'unico hidden layer con un dropout del 0.3%. Infine l'output layer composto da 4 neuroni. Il modello presenta un learning rate del 0.01%, dunque ad ogni aggiornamento, i pesi della rete vengono modificati in misura pari al 1% del gradiente calcolato. L'ottimizzatore e la loss function utilizzati sono rispettivamente l'SGD e la *Cross Entropy Loss*.

Tabella 3.6. Architettura della rete neurale

Layer	Dettagli
Input Layer	41 neuroni
Hidden Layer	59 neuroni, Attivazione: Tanh
Dropout	p = 0.3
Output Layer	4 neuroni

Il modello è stato allenato con un early stop di 10 epoche, fermandosi alla 112ª epoca. La rete neurale nel processo di validazione ottiene: *Accuracy*: 86%, *Recall*:

88% e *F1-score*: 86%. Le statistiche sono influenzate negativamente dalla classe minoritaria R2L, notevole il recall sulla classe probe del 95%. Il componente è stato implementato in Python tramite la libreria *pytorch*.

3.3.5 Combinazione dei componenti

L'ensemble, come da definizione, è un approccio che si basa sull'aggregare più modelli per migliorare le previsioni dei singoli. Di seguito propongo il mio metodo per effettuare la previsione finale basandomi su quelle effettuate dai vari componenti:

Divido il processo di previsione in due parti, partendo dall'approccio **signature-based**, che sfrutta le probabilità fornite dai modelli SVM, MARS e ANN per ciascuna classe di attacco.

- **Classi "Normal" e "Probe"**: la probabilità è ottenuta calcolando la media delle probabilità stimate dai tre modelli, assegnando un peso pari a $\frac{1}{3}$ a ciascun contributo:
$$P_{\text{normal,probe}} = \frac{1}{3}P_{\text{SVM}} + \frac{1}{3}P_{\text{MARS}} + \frac{1}{3}P_{\text{ANN}} \tag{3.25}$$
- **Classe "R2L"**: la previsione per questa classe viene calcolata combinando solo SVM e MARS, escludendo ANN. La probabilità è data da:
$$P_{\text{R2L}} = \frac{1}{2}P_{\text{SVM}} + \frac{1}{2}P_{\text{MARS}} \tag{3.26}$$
Questo perché i modelli SVM e MARS hanno dimostrato una maggiore capacità di rilevare gli attacchi R2L, mentre ANN tende a soffrire eccessivamente lo sbilanciamento della classe.
- **Classe "DoS"**: la previsione è ottenuta selezionando la **massima probabilità** tra i tre modelli:
$$P_{\text{DoS}} = \max(P_{\text{SVM}}, P_{\text{MARS}}, P_{\text{ANN}}) \tag{3.27}$$
- **Previsione dell'ensemble:** la previsione finale dell'ensemble è data da:
$$P_{\text{Finale}} = \max(P_{\text{DoS}}, P_{\text{probe}}, P_{\text{normal}}, P_{\text{R2L}}) \tag{3.28}$$

Per identificare le anomalie (approccio **anomaly-detection**), il sistema sfrutta le probabilità di appartenenza alla classe *normal*, fornite dai modelli MARS e SVM. La probabilità di anomalia per ogni istanza è calcolata come complemento della probabilità normale, ovvero:

$$P_{\text{anomalia},MARS} = 1 - P_{\text{normal},MARS} \tag{3.29}$$
$$P_{\text{anomalia},SVM} = 1 - P_{\text{normal},SVM} \tag{3.30}$$

Successivamente, si combina il contributo di entrambi i modelli con una media pesata, ottenendo la stima finale:

$$P_{\text{anomalia}} = \frac{1}{2}P_{\text{anomalia},MARS} + \frac{1}{2}P_{\text{anomalia},SVM} \tag{3.31}$$

La classificazione finale viene determinata confrontando la probabilità di anomalia con una soglia configurabile T, tale che:

$$\hat{y} = \begin{cases} \text{anomaly}, & \text{se } P_{\text{anomalia}} > T \\ \text{normal}, & \text{altrimenti} \end{cases} \tag{3.32}$$

Classificazione finale

L'ultima fase del processo di classificazione prevede la combinazione delle due parti. Il sistema segue il seguente schema decisionale:

1. I dati in ingresso vengono prima analizzati dal multiclassificatore **signature-based**, il quale assegna a ciascuna istanza una delle classi definite: *Normal*, *DoS*, *Probe*, o *R2L*.

2. Le istanze classificate come appartenenti alle classi di attacco (*DoS*, *Probe*, *R2L*) vengono immediatamente etichettate come **anomalia**, senza necessità di ulteriori verifiche.

3. Le istanze classificate come *Normal* dal multiclassificatore vengono ulteriormente analizzate dalla componente **anomaly-based**, che calcola la probabilità di anomalia in base ai modelli MARS e SVM.

4. Se la probabilità di anomalia dell'istanza supera la soglia T, essa viene riclassificata come **anomalia**, altrimenti viene mantenuta come **normale**.

L'algoritmo finale può essere formalizzato come segue:

$$\hat{y} = \begin{cases} \text{anomaly}, & \text{se } P_{\text{Finale}} \neq P_{\text{normal}} \\ \text{anomaly}, & \text{se } P_{\text{anomalia}} > T \text{ e } P_{\text{Finale}} = P_{\text{normal}} \\ \text{normal}, & \text{altrimenti} \end{cases} \tag{3.33}$$

L'integrazione dei due approcci permette di sfruttare la precisione della componente **signature-based** per gli attacchi conosciuti e la flessibilità dell'**anomaly-based** per l'individuazione di nuove minacce. Dunque, la seguente strategia, consente di massimizzare l'affidabilità della classificazione, riducendo il numero di falsi positivi e garantendo un'elevata capacità di rilevamento.

Capitolo 4

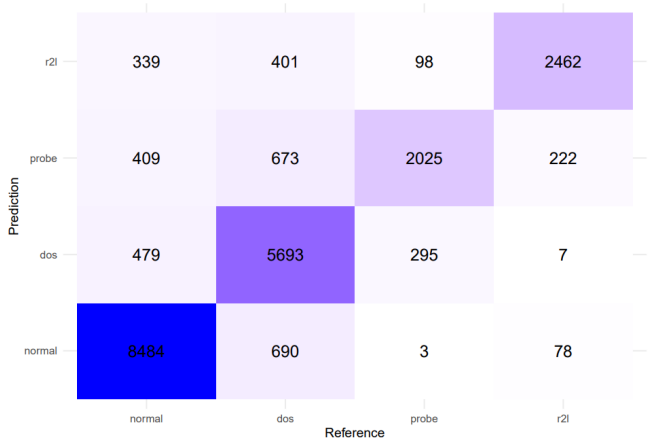
Esperimenti

Il seguente capitolo espone i risultati ottenuti dai vari componenti e dall'ensemble completo, enfatizzando come la previsione combinata superi quella dei singoli. Tra questi è presente una simulazione di attacco zero-day, effettuata per testare la robustezza dell'approccio anomaly-based.

4.1 Risultati Signature-Detection

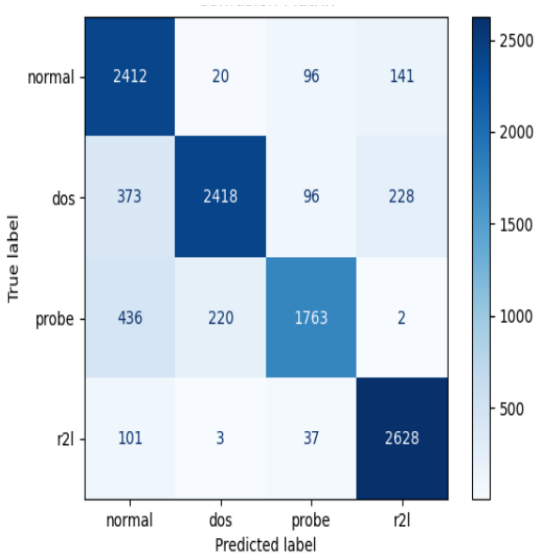
I risultati ottenuti sono stati effettuati sul test set NSL-KDD99

Componente MARS



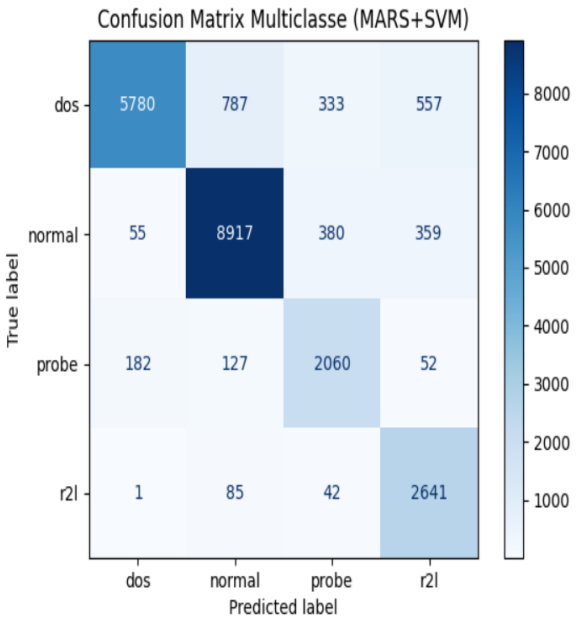
Classe	TP	FP	FN	Accuracy	Precision	Recall	F1 Score
Normal	8484	771	1227	0.911	0.917	0.874	0.894
DoS	5693	781	1764	0.886	0.879	0.762	0.816
Probe	2025	1304	396	0.924	0.608	0.836	0.703
R2L	2462	838	307	0.948	0.746	0.889	0.811
Globale				0.840	0.768	0.840	0.810

Componente SVM



Classe	TP	FP	FN	Accuracy	Precision	Recall	F1
Normal	2412	910	257	0.894	0.726	0.904	0.805
DoS	2418	243	697	0.915	0.909	0.776	0.837
Probe	1763	229	658	0.919	0.885	0.728	0.801
R2L	2628	371	141	0.953	0.876	0.950	0.910
Globale				0.847	0.849	0.840	0.838

Ensemble SVM + MARS



Classe	TP	FP	FN	Accuracy	Precision	Recall	F1
dos	5780	238	1677	0.914	0.96	0.775	0.86
normal	8917	999	794	0.920	0.90	0.918	0.91
probe	2060	755	361	0.950	0.73	0.851	0.79
r2l	2641	968	128	0.951	0.73	0.954	0.83
Globale				0.870	0.836	0.874	0.850

Ensemble completo

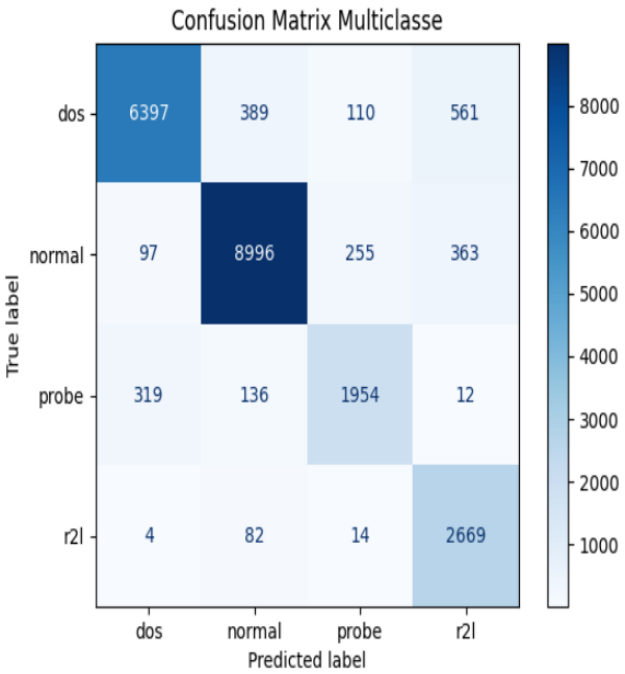
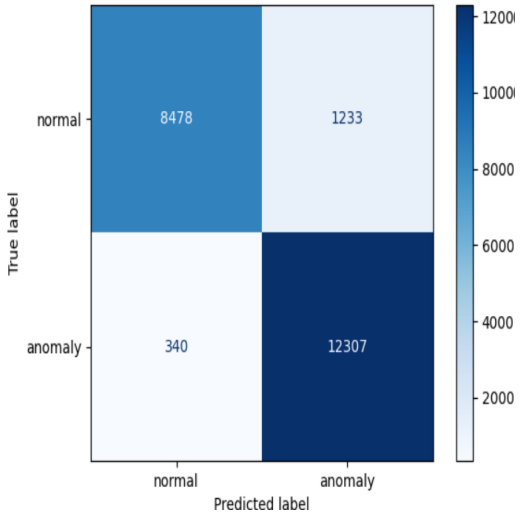


Figura 4.1. Risultati unione ANN,MARS e SVM

Classe	TP	FP	FN	Accuracy	Precision	Recall	F1
dos	6397	420	1060	0.934	0.938	0.858	0.896
normal	8996	607	715	0.938	0.937	0.926	0.932
probe	1954	379	467	0.964	0.838	0.807	0.822
r2l	2669	936	100	0.954	0.740	0.964	0.838
Globale				0.900	0.870	0.900	0.882

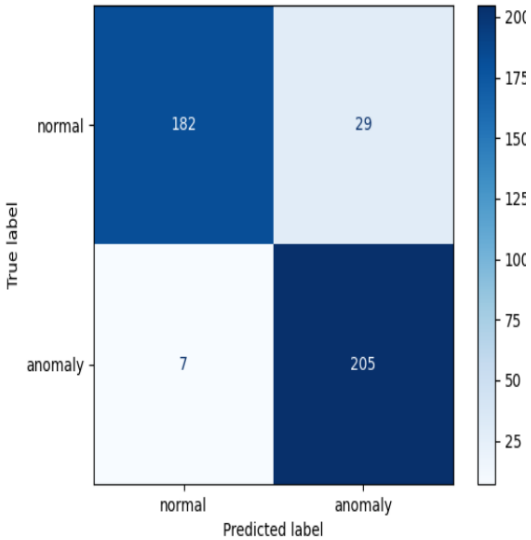
4.2 Risultati Anomaly-Detection

Il primo test è stato effettuato con il NSL-KDD99 test set per osservare il comportamento con dati presenti in allenamento.



Metrica	Valore
Tasso di Falsi Positivi	12.7%
Tasso di Rilevazione	97.3%

L'ultimo test è stato eseguito su un dataset formato da 50% di traffico normale e dal 50% di traffico malevolo U2R. Testiamo la robustezza dell'ensemble agli attacchi non presenti nel training (simulazione attacchi zero-day).

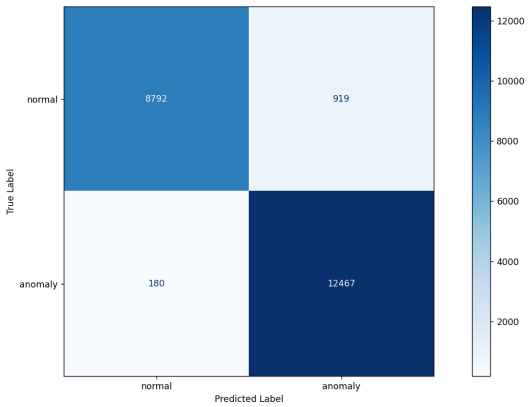


Metrica	Valore
Tasso di Falsi Positivi	13.0%
Tasso di Rilevazione	96.7%

4.3 Modelli in serie

Per testare come cambia il modello combinando i due approcci è stato utilizzato il test set NSL-KDD99. I risultati della multi-classificazione (signature-based) sono rappresentati dalla Figura 4.1 con le rispettive metriche.

I dati, prima classificati normali, sono rianalizzati venendo classificati come anomali



Metrica	Valore
Tasso di Falsi Positivi	9%
Tasso di Rilevazione	98.6%

se non rientrano nella soglia preconfigurata. I dati prima classsificati come una delle classi malevole, sono già etichettate come anomalie, dunque non vengono rianalizzati.

Capitolo 5

Conclusioni

Gli **Intrusion Detection Systems** rappresentano un elemento fondamentale per la protezione delle infrastrutture informatiche, consentendo di identificare e mitigare potenziali minacce prima che queste possano comprometterle. In questo lavoro è stato sviluppato un approccio basato su un **ensemble** di modelli di *Machine Learning*, aggregando le due politiche principali di rilevazione intrusioni al fine di combinarne i vantaggi.

Per quanto riguarda la componente **signature-based**, i risultati sperimentali evidenziano come l'ensemble superi le prestazioni dei singoli modelli, garantendo una classificazione più accurata degli attacchi. Il sistema è infatti in grado di identificare correttamente il **90%** degli esempi, raggiungendo un tasso di rilevazione del **90%**.

Un aspetto particolarmente rilevante emerge dall'analisi degli attacchi *User-to-Root*, notoriamente difficili da rilevare a causa della loro bassa rappresentazione nei dataset. I test condotti dimostrano che l'ensemble proposto mantiene una **robustezza elevata** nei confronti di attacchi *zero-day*, riuscendo a rilevare il **97%** delle anomalie, con un tasso di falsi positivi pari al **12%**.

L'implementazione di una pipeline di classificazione con modelli in serie ha ulteriormente affinato le prestazioni del sistema. In particolare, questa strategia ha permesso di ridurre il tasso di falsi positivi al **9%**, migliorando al contempo il tasso di rilevazione fino a raggiungere il **98.6%**.

Questi risultati confermano l'efficacia dell'approccio adottato e sottolineano l'importanza dell'integrazione di tecniche *signature-based* e *anomaly-based* per il rilevamento delle intrusioni. Il sistema proposto rappresenta una soluzione promettente per la sicurezza delle reti, fornendo un metodo scalabile e adattabile in grado di individuare con precisione sia attacchi noti che minacce emergenti.

Sviluppi futuri: Nonostante i risultati ottenuti, esistono margini di miglioramento che possono essere esplorati in futuri sviluppi. Tra questi, si evidenzia l'ottimizzazione della gestione degli squilibri nelle classi, l'integrazione di ulteriori modelli di deep learning e l'adozione di strategie di aggiornamento continuo per migliorare la capacità del sistema di adattarsi a nuove minacce in tempo reale.

Bibliografia

[1] UCI KDD Archive. *KDD Cup 1999 Data*. 1999. URL: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[2] Bradley Boehmke. *HOML: Mars*. https://bradleyboehmke.github.io/HOML/mars.html.

[3] Muataz Salam Al-Daweri et al. "An Analysis of the KDD99 and UNSW-NB15 Datasets for the Intrusion Detection System". In: *Symmetry* 12 (2020), p. 1666. DOI: doi:10.3390/sym12101666.

[4] Jesús Díaz-Verdejo et al. "On the Detection Capabilities of Signature-Based Intrusion Detection Systems in the Context of Web Attacks". In: *Applied Sciences* 12.2 (2022), p. 852. DOI: 10.3390/app12020852. URL: https://doi.org/10.3390/app12020852.

[5] MIT Lincoln Laboratory. *1998 DARPA Intrusion Detection Evaluation Dataset*. 1998. URL: https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset.

[6] Preeti Mishra et al. "A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection". In: *IEEE Communications surveys & tutorials* 21.1 (2019), pp. 686–723. DOI: 10.1109/comst.2018.2847722.

[7] Srinivas Mukkamala, Andrew H. Sung e Ajith Abraham. "Intrusion detection using an ensemble of intelligent paradigms". In: *Journal of Network and Computer Applications* 28.2 (2005). Computational Intelligence on the Internet, pp. 167–182. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2004.01.003. URL: https://www.sciencedirect.com/science/article/pii/S1084804504000049.

[8] University of New Brunswick (NSL-KDD Project). *NSL-KDD DataSet*. 2009. URL: https://web.archive.org/web/20150205070216/http://nsl.cs.unb.ca/NSL-KDD/.