Previsione dello Stato delle Connessioni di Rete

Componenti del gruppo:

Maria Forte, [MAT. 758345], m.forte33@studenti.uniba.it

Link GitHub:

https://github.com/mariaforte02/Progetto_Icon.git

A.A 2024-2025

Sommario

Capitolo 0) Introduzione

Obiettivo del Progetto

L'obiettivo principale di questo progetto è sviluppare un sistema di analisi predittiva e di interrogazione basato su un dataset contenente informazioni dettagliate sulle connessioni di rete, con lo scopo di identificare i fattori chiave che influenzano il comportamento delle connessioni in un ambiente di rete simulato.

Il sistema consentirà agli utenti di:

1. **Eseguire interrogazioni mirate (query)**: Gli utenti potranno esplorare dettagli specifici delle connessioni, come la durata, il tipo di protocollo, il servizio utilizzato, la quantità di byte inviati e ricevuti, e altre caratteristiche tecniche.
2. **Effettuare previsioni**: Utilizzando modelli di Machine Learning, il sistema sarà in grado di prevedere determinati risultati o eventi, come l'identificazione di anomalie o la probabilità che una connessione rappresenti un comportamento anomalo.

Requisiti funzionali

Il progetto è stato realizzato in Python, un linguaggio che offre diverse librerie utili che ci permettono di trattare dati in modo facile ed intuitivo.

La versione di Python utilizzata è: 3.12.

L'IDE utilizzato è: IntelliJ

Le librerie utilizzate sono:

- **pandas** - Libreria per la manipolazione e l'analisi dei dati.
- **matplotlib.pyplot** - Modulo per creare grafici e visualizzazioni.
- **sklearn.model_selection** - Modulo per suddividere dataset e validazione incrociata.
- **sklearn.ensemble** - Implementa metodi ensemble come Random Forest.
- **sklearn.tree** - Costruisce alberi decisionali per classificazione e regressione.
- **sklearn.metrics** - Calcola metriche di valutazione dei modelli.
- **sklearn.neighbors** - Implementa il classificatore K-Nearest Neighbors (KNN).

- **sklearn.svm** - Implementa Support Vector Machines (SVM).
- **kneed** - Libreria per trovare il "gomito" in curve, utile per determinare il numero ottimale di cluster.
- **owlready2** - Libreria per lavorare con ontologie OWL (Web Ontology Language) in Python.
- **sklearn.preprocessing** - Modulo per la pre-elaborazione dei dati, come la normalizzazione e la codifica.
- **sklearn**.**cluster** – Libreria utilizzata per implementare algoritmi di clustering, un'area dell'apprendimento automatico non supervisionato.
- **seaborn** - Libreria per la visualizzazione dei dati basata su matplotlib, con un'interfaccia di alto livello.

Installazione e avvio

Per prima cosa bisogna aprire la cartella del progetto all'interno del nostro IDE, nel codice saranno presenti delle istruzioni commentate che possono essere eseguite solo una volta (come l'installazione dei requisiti) ed infine bisogna avviare il programma a partire dal file contenuto nella cartella denominato "main.py".

Capitolo 1) Creazione del dataset

Il dataset utilizzato in questo progetto è stato costruito partendo da un file contenente informazioni dettagliate sulle connessioni di rete. Per garantire che i dati fossero adatti all'analisi e al machine learning, è stato effettuato un processo di pulizia e preprocessing attraverso uno script Python dedicato.

Le operazioni principali svolte includono:

1. **Rimozione di valori indesiderati:** Sono stati eliminati record con valori mancanti in colonne essenziali, come protocol_type, service, src_bytes, dst_bytes e duration.
2. **Rimozione di duplicati:** Per garantire l'unicità dei dati, sono state eliminate le righe duplicate.
3. **Codifica delle variabili categoriche:** Le colonne testuali, come protocol_type, service e flag, sono state convertite in valori numerici tramite LabelEncoder.
4. **Standardizzazione delle variabili numeriche:** Le colonne contenenti valori numerici, come duration, src_bytes e dst_bytes, sono state scalate utilizzando lo standard scaler per uniformare le scale.

   Al termine di queste operazioni, il dataset raffinato è stato salvato in un file CSV pronto per le analisi successive.

Caratteristiche del Dataset

Il dataset finale contiene 22.542 righe e 42 colonne, tra cui:

1. **duration:** Durata della connessione.
2. **protocol_type:** Tipo di protocollo utilizzato (ad esempio, TCP, UDP, ICMP).
3. **service:** Servizio richiesto (ad esempio, HTTP, FTP, SMTP).
4. **flag:** Indicatore dello stato della connessione.
5. **src_bytes:** Numero di byte inviati dalla sorgente.
6. **dst_bytes:** Numero di byte ricevuti dalla destinazione.
7. **wrong_fragment:** Numero di frammenti errati rilevati.
8. **urgent:** Numero di pacchetti urgenti inviati.
9. **date:** Data fittizia generata per ogni record.

10. **Altri indicatori:** Variabili come tassi di errore (serror_rate, srv_rerror_rate), numero di accessi non autorizzati e metriche relative alle prestazioni.

Preprocessing del Dataset

Lo script di preprocessing automatizza il processo di pulizia e trasformazione del dataset, seguendo queste fasi principali:

1. **Caricamento e rimozione di valori nulli:**
    - Il dataset viene caricato dal file CSV.
    - Le righe contenenti valori nulli in colonne essenziali sono eliminate.
2. **Codifica delle variabili categoriche:**
    - Colonne testuali come protocol_type e service sono state trasformate in valori numerici per renderle compatibili con gli algoritmi di machine learning.
3. **Standardizzazione delle variabili numeriche:**
    - Variabili come duration, src_bytes e dst_bytes sono state scalate per garantire una distribuzione uniforme e comparabile.
4. **Rimozione di duplicati:**
    - Le righe duplicate sono state eliminate per garantire l'integrità dei dati.
5. **Salvataggio del dataset raffinato:**
    - Il dataset pre-elaborato è stato salvato in un file CSV pronto per essere utilizzato in analisi avanzate.

Obiettivo del Preprocessing

L'obiettivo di queste operazioni è garantire che il dataset sia:

- **Pronto per l'analisi esplorativa:** Attraverso visualizzazioni e statistiche descrittive.
- **Adatto al machine learning:** Grazie alla pulizia, codifica e standardizzazione dei dati.
- **Strutturato per query e analisi:** Facilitando l'estrazione di informazioni rilevanti sulle connessioni di rete.

Capitolo 2) Analisi dei dati

L’analisi preliminare dei dati rappresenta un passaggio fondamentale per il successo del processo di apprendimento supervisionato e la formulazione di query precise. Questa fase permette di comprendere meglio le informazioni contenute nel file **"Test_data_cleaned_unified.csv"** attraverso una serie di visualizzazioni grafiche.

I grafici generati con la libreria **matplotlib** si sono rivelati strumenti essenziali per:

- Identificare tendenze e anomalie.
- Individuare correlazioni tra variabili.
- Effettuare operazioni di pulizia e conversione dei dati per migliorare la qualità del dataset.
- Generare ipotesi basate su dati chiari e strutturati.

Questa esplorazione visiva è essenziale per garantire che i dati siano adeguatamente preparati per l’apprendimento automatico, assicurando che i modelli supervisionati siano addestrati con dati affidabili e pertinenti.

Le funzioni usate da noi prese dalla libreria, matplotlib e servono a:

1. plt.figure(figsize=(x, y)): Questa funzione crea una nuova figura con dimensioni specificate. figsize definisce le dimensioni della figura (in pollici), dove x è la larghezza e y è l'altezza. Questo è utile quando si desidera controllare la dimensione del grafico.
2. plt.scatter(df['AgeMonths'], df['WeightKg'], alpha=0.7, c='purple'): Genera un grafico a dispersione utilizzando i dati contenuti nelle colonne AgeMonths e WeightKg del DataFrame df. I punti del grafico sono semi-trasparenti (con alpha=0.7) e di colore viola (c='purple').

3 . plt.pie(): Crea un grafico a torta. Mostra la ripartizione percentuale delle adozioni per tipologia con etichette, colori e percentuali.

4. plt.bar(): Crea un grafico a barre. Utilizzato per visualizzare la distribuzione degli animali per probabilità di adozione. Le barre sovrapposte mostrano la probabilità di essere adottati o meno con differenti colori.

5. plt.xlabel()`, `plt.ylabel(): Assegna etichette agli assi x e y rispettivamente.

6. plt.title(): Imposta il titolo del grafico.

7. plt.grid(True): Aggiunge una griglia per facilitare la lettura del grafico.

8. plt.show(): Visualizza il grafico creato.

I grafici creati usando le funzioni descritte in precedenza sono:

Grafico 1: Distribuzione delle Connessioni per Tipologia (Grafico a Torta)

Questo grafico a torta rappresenta la ripartizione percentuale delle diverse tipologie di connessioni (es. protocol_type). Fornisce un'idea della predominanza di determinate categorie rispetto ad altre, evidenziando eventuali squilibri nel dataset.

Codice per il grafico a torta:

```
# Grafico a torta con nomi dei protocolli come etichette
plt.figure(figsize=(8, 8))
counts.plot.pie(
    autopct='%1.1f%%',
    labels=counts.index,  # Aggiunge i nomi come etichette
    colors=['gold', 'lightblue', 'pink', 'green'],
    startangle=90
)

plt.title('Distribuzione delle Connessioni per Tipologia')
plt.ylabel('')  # Rimuove l'etichetta dell'asse Y
plt.show()
```

Grafico 2: Distribuzione del Tasso di Errori (Grafico a Barre)

Questo grafico a barre rappresenta la distribuzione del tasso di errori (serror_rate), suddiviso in intervalli. La visualizzazione consente di individuare eventuali concentrazioni di connessioni con specifici livelli di errori, offrendo una panoramica chiara della presenza di connessioni con errori elevati o bassi. Questo tipo di analisi aiuta a identificare comportamenti anomali o regolarità nel dataset, supportando ulteriori analisi per il rilevamento di intrusioni.

Codice per il grafico a barre:

```
# Creazione del grafico a barre
plt.figure(figsize=(10, 6))
serror_rate_distribution.plot.bar(color='skyblue')
plt.title('Distribuzione del Tasso di Errori (serror_rate)')
plt.xlabel('Intervalli di Tasso di Errori S')
plt.ylabel('Numero di Connessioni')
plt.show()
```

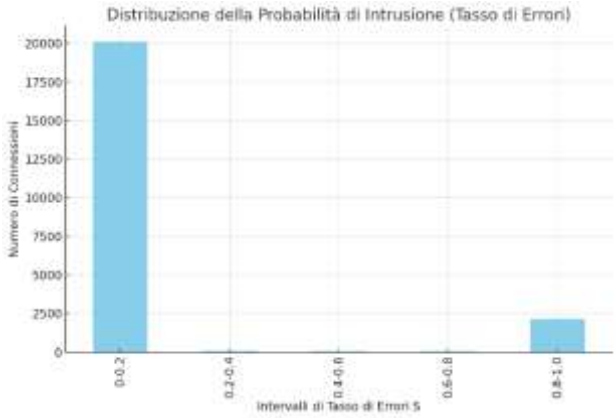


Grafico 3: Byte Inviati rispetto ai Byte Ricevuti (Grafico a Dispersione)

Questo grafico a dispersione analizza la relazione tra i byte inviati (src_bytes) e i byte ricevuti (dst_bytes). Evidenzia eventuali cluster o pattern specifici che potrebbero indicare connessioni anomale.

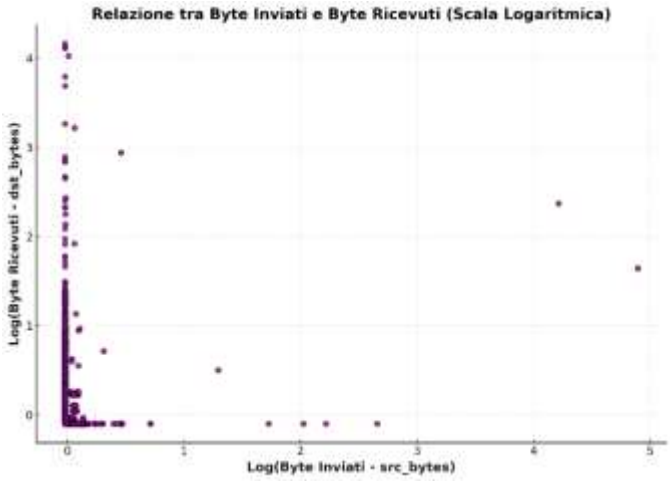
Codice per il grafico a dispersione:

```
# Creazione del grafico
plt.figure(figsize=(12, 8))
plt.scatter(df['log_src_bytes'], df['log_dst_bytes'], alpha=0.6, color='purple')

# Titolo e etichette degli assi
plt.title('Relazione tra Byte Inviati e Byte Ricevuti (Scala Logaritmica)')
plt.xlabel('Log(Byte Inviati - src_bytes)')
plt.ylabel('Log(Byte Ricevuti - dst_bytes)')

# Aggiunta di una griglia
plt.grid(True, linestyle='--', alpha=0.4)

# Visualizzazione del grafico
plt.show()
```



Questi grafici non solo offrono una visione chiara e dettagliata dei dati, ma forniscono anche spunti utili per ulteriori analisi e per migliorare la qualità dei modelli di apprendimento supervisionato.

Capitolo 3) Apprendimento Supervisionato

L’apprendimento supervisionato rappresenta una delle tecniche cardine nel dominio del machine learning, in cui un modello computazionale viene istruito su un dataset etichettato, ossia un insieme di dati in cui le risposte corrette (etichette) sono previamente note. L’obiettivo primario di questa metodologia consiste nel costruire un sistema in grado di effettuare previsioni accurate su dati mai analizzati in precedenza, attraverso un processo di generalizzazione basato sulle informazioni intrinseche al dataset di addestramento.

Nel corso del processo di apprendimento supervisionato, il modello apprende una mappatura funzionale tra le variabili indipendenti (input) e i corrispondenti risultati attesi (output), avvalendosi di algoritmi di ottimizzazione che incrementano progressivamente la sua capacità predittiva. Questo approccio trova applicazione in un ampio spettro di contesti, tra cui la classificazione di immagini, il riconoscimento vocale e la previsione di fenomeni o tendenze.

Nel contesto del nostro progetto, abbiamo deciso di implementare quattro modelli di apprendimento supervisionato, ciascuno caratterizzato da peculiarità metodologiche distintive:

- **Decision Tree**: Un classificatore che adotta una struttura gerarchica ad albero, in cui le foglie rappresentano le classi di appartenenza (o le probabilità di appartenenza), mentre i nodi e la radice corrispondono a condizioni definite sulle caratteristiche di input. L’algoritmo determina il percorso ottimale nell’albero sulla base della valutazione delle condizioni poste nei nodi, conducendo infine a una previsione deterministica.
- **Random Forest**: Una tecnica di apprendimento per insiemi (ensemble learning) basata sul metodo del *bagging*, che aggrega le predizioni di una collezione di alberi decisionali. Ogni albero fornisce una stima indipendente e la risposta finale viene ottenuta tramite la media (per problemi di regressione) o il voto di maggioranza (per problemi di classificazione), migliorando così la robustezza e riducendo il rischio di overfitting rispetto a un singolo albero.
- **K-Nearest Neighbors (KNN)**: Un algoritmo non parametrico che classifica un'istanza sulla base dei suoi *k* vicini più prossimi nel medesimo spazio delle caratteristiche. Nel caso della classificazione, l’etichetta viene assegnata considerando la classe più frequente tra i vicini; per la regressione, la previsione è calcolata come la media dei valori dei vicini selezionati.

- **Support Vector Machine (SVM)**: Un potente algoritmo supervisionato, particolarmente efficiente nella classificazione. L'SVM individua un iperpiano ottimale che separa le classi di appartenenza massimizzando il margine tra i punti più prossimi di ciascuna classe (*support vectors*). Inoltre, grazie all'uso di funzioni kernel, l'SVM è in grado di trattare dati non linearmente separabili, mappandoli in uno spazio di dimensioni superiori dove la separazione diventa più agevole.

DECISION TREE

Librerie Utilizzate:

- **pandas:**Per la manipolazione e l'analisi dei dati, come il caricamento del dataset e la gestione delle tabelle. In questo caso, pd.read_csv(file_path) carica il dataset relativo alla rilevazione delle intrusioni di rete, che successivamente viene analizzato e filtrato.
- **matplotlib.pyplot:** Utilizzata per creare visualizzazioni, in particolare per rappresentare graficamente l'albero decisionale. La funzione plt.show() consente di visualizzare grafici come l'albero decisionale generato dall'analisi delle intrusioni di rete.
- **sklearn.model_selection:** Questa libreria offre strumenti per la suddivisione dei dati relativi alle connessioni di rete in set di addestramento e di test, tramite la funzione train_test_split(), e per la ricerca degli iperparametri ottimali con GridSearchCV.
- **sklearn.tree:** Fornisce l'implementazione del classificatore ad albero decisionale, DecisionTreeClassifier, e la funzione plot_tree() per visualizzare l'albero decisionale derivato dall'analisi delle connessioni di rete.
- **sklearn.metrics:** Utilizzata per calcolare l'accuratezza del modello nella classificazione delle intrusioni di rete tramite la funzione accuracy_score().

Funzioni della Classe DecisionTree (Adattata al tuo dataset)

1. **__init__:**
    - Carica il dataset dal file CSV e ne verifica il contenuto. Stampa la distribuzione delle classi (colonna flag), utile per comprendere come sono suddivise le categorie. Inoltre, elimina le classi con meno di 2 campioni per evitare di includere categorie non rappresentative nei dati.

2. **preprocess_data:**
    - Estrae le caratteristiche principali e la variabile target dal dataset. Le caratteristiche selezionate sono duration, src_bytes e dst_bytes, che rappresentano rispettivamente la durata della connessione, i byte inviati dalla sorgente e i byte ricevuti dalla destinazione. La variabile target è flag, che identifica se una connessione è legittima o anomala. Questa funzione prepara i dati per il training del modello.
3. **optimize_hyperparameters:**
    - Suddivide i dati in training set e test set e ottimizza gli iperparametri tramite GridSearchCV. Gli iperparametri testati includono il criterio di suddivisione (gini o entropy), la profondità massima dell'albero, il numero minimo di campioni richiesti per una suddivisione e il numero minimo di campioni per ogni foglia. La funzione identifica i migliori parametri per massimizzare l'accuratezza del modello.
4. **train_model:**
    - Addestra il modello di albero decisionale utilizzando i dati di addestramento e valuta la sua performance sui dati di test. Calcola l'accuratezza delle predizioni, fornendo una misura chiave della capacità del modello di classificare correttamente le connessioni.
5. **plot_best_params_table:**
    - Dopo la ricerca degli iperparametri, crea una tabella che riassume i migliori parametri trovati. Utilizzando matplotlib, la funzione genera una visualizzazione chiara e ben formattata per presentare i risultati della ricerca.
6. **plot_decision_tree:**
    - Visualizza graficamente l'albero decisionale generato dal modello. Utilizza la funzione plot_tree() di sklearn.tree per rappresentare l'albero, con la possibilità di limitare la profondità per evitare complessità eccessive. Il grafico mostra come il modello prende decisioni basandosi sulle caratteristiche duration, src_bytes e dst_bytes.

Nel contesto della rilevazione delle intrusioni di rete, l'algoritmo Decision Tree si rivela particolarmente utile per costruire un modello interpretabile, capace di classificare le connessioni come legittime o anomale. Le variabili principali (duration, src_bytes e dst_bytes) forniscono informazioni essenziali per distinguere tra tipi di connessione. L'algoritmo riesce a gestire queste variabili numeriche senza richiedere trasformazioni complesse, rendendolo ideale per un'applicazione pratica nella sicurezza di rete. La sua capacità di essere facilmente visualizzato come un albero decisionale consente di spiegare il processo decisionale anche a persone non esperte in machine learning.

Ecco una descrizione dettagliata basata sul tuo dataset di rilevazione delle intrusioni di rete, seguendo il modello fornito:

SWM (Support Vector Machine)

Librerie Usate

- **pandas**: Utilizzata per la gestione e la manipolazione dei dati. Viene impiegata per caricare il dataset e trasformarlo in un formato gestibile (DataFrame), facilitando le operazioni di analisi e preprocessing.
- **scikit-learn**:
    - **train_test_split**: Per suddividere il dataset in due parti: un set di addestramento (training set) e un set di test (test set), garantendo una separazione dei dati per l'addestramento e la valutazione del modello.
    - **SVC**: L'algoritmo implementa la Support Vector Machine (SVM), utilizzato per creare il modello di classificazione. Il kernel usato è "lineare", ma è possibile scegliere anche altre opzioni, come rbf o poly, per adattarsi a dati non lineari.
    - **accuracy_score**: Per calcolare l'accuratezza del modello, confrontando le predizioni con i valori reali del dataset.
    - **LabelEncoder**: Per convertire le variabili categoriche in valori numerici, consentendo al modello SVM di elaborarle correttamente.
    - **StandardScaler**: Per normalizzare le variabili numeriche, evitando che differenze di scala tra le caratteristiche influenzino negativamente le prestazioni del modello.

Funzioni della Classe SVMPacketAnalysis

a) __init__(self, data_path)

- Questa funzione è il costruttore della classe. Viene utilizzata per caricare il dataset dal percorso fornito (data_path) e inizializzare il modello SVM con un kernel lineare.

```
class SVMPacketAnalysis:
    def __init__(self, data_path):
        # Carica il dataset
        self.dataset = pd.read_csv(data_path)
        self.model = SVC(kernel='linear')  # Puoi modificare il kernel a 'rbf', 'poly', ecc.
```

b) preprocess_data(self)

- Questa funzione gestisce la pre-elaborazione dei dati:

    1. **Codifica delle variabili categoriche**: Variabili come protocol_type e service vengono convertite in valori numerici utilizzando LabelEncoder, così da essere compatibili con il modello SVM.
    2. **Normalizzazione delle variabili numeriche**: Variabili come duration, src_bytes, dst_bytes, wrong_fragment e urgent vengono scalate utilizzando StandardScaler, per evitare che scale diverse tra le caratteristiche influenzino in modo sproporzionato il modello.
    3. **Selezione delle colonne di interesse**: Le caratteristiche (X) includono variabili rilevanti per la classificazione, mentre la variabile target (y) è la colonna flag, che identifica se una connessione è legittima o rappresenta un'anomalia.

```
def preprocess_data(self):
    # Codifica variabili categoriali (ad esempio, protocol_type, service)
    label_encoders = {}
    categorical_columns = ["protocol_type", "service"]

    for col in categorical_columns:
        le = LabelEncoder()
        self.dataset[col] = le.fit_transform(self.dataset[col])
        label_encoders[col] = le

    # Normalizza le variabili numeriche
    scaler = StandardScaler()
    numeric_columns = ["duration", "src_bytes", "dst_bytes", "wrong_fragment", "urgent"]
    self.dataset[numeric_columns] = scaler.fit_transform(self.dataset[numeric_columns])

    # Seleziona le colonne desiderate per le feature (X)
    X = self.dataset[["duration", "src_bytes", "dst_bytes", "wrong_fragment", "urgent", "protocol_type", "service"]]

    # Definisci la colonna target (y)
    y = self.dataset["flag"]  # La variabile target, che identifica se una connessione è normale o anomala

    return X, y
```

c) train_model(self)

- Questa funzione addestra e valuta il modello:
    1. **Suddivisione dei dati**: Il dataset viene suddiviso in training set e test set tramite train_test_split, per garantire una valutazione affidabile.
    2. **Addestramento del modello**: Il modello SVM viene addestrato sul training set utilizzando il metodo fit.
    3. **Predizioni**: Il modello effettua previsioni sul test set tramite il metodo predict.
    4. **Valutazione**: L'accuratezza del modello viene calcolata confrontando le predizioni con i valori reali tramite accuracy_score.

```
def train_model(self):
    X, y = self.preprocess_data()

    # Suddividi i dati in training e test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Addestra il modello
    self.model.fit(X_train, y_train)

    # Fai previsioni e calcola l'accuratezza
    y_pred = self.model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    print("Accuracy:", accuracy)
```

Motivazione dell'Utilizzo dell'Algoritmo SVM

Nel contesto della rilevazione delle intrusioni di rete, l'algoritmo SVM è stato scelto per la sua capacità di classificare efficacemente connessioni legittime e anomalie. L'SVM cerca un iperpiano che separi le classi nel miglior modo possibile, massimizzando il margine tra le connessioni normali e quelle anomale.

L'algoritmo è particolarmente utile in scenari come questo, dove sono presenti sia variabili numeriche (come duration e src_bytes) che categoriali (come protocol_type e service). La flessibilità dell'SVM nel gestire dati lineari e non lineari lo rende una scelta robusta per problemi di classificazione. Nel nostro caso, è stato scelto un kernel lineare per semplicità e per osservare il comportamento del modello su dati già rappresentabili linearmente.

Conclusione

L'approccio SVM applicato al dataset di rilevazione delle intrusioni di rete consente di costruire un modello robusto, capace di classificare connessioni come legittime o anomale in modo affidabile. Il processo include la pre-elaborazione dei dati (codifica

delle variabili categoriali e normalizzazione delle variabili numeriche), la suddivisione in training e test set, l'addestramento del modello e la sua valutazione. Questo flusso ben strutturato garantisce un'analisi completa e una valutazione affidabile delle prestazioni del modello. L'SVM si rivela uno strumento potente per migliorare la sicurezza delle reti, rilevando rapidamente potenziali intrusioni o anomalie.

KNN

Librerie Utilizzate

Le librerie principali utilizzate nel file sono:

Librerie Usate

- **pandas**: Utilizzata per la gestione e la manipolazione dei dati. Viene impiegata per caricare il dataset e trasformarlo in un formato gestibile (DataFrame), facilitando le operazioni di analisi e preprocessing.
- **scikit-learn**:
    - **train_test_split:** Per suddividere il dataset in due parti: un set di addestramento (training set) e un set di test (test set). Questa funzione garantisce una separazione dei dati per l'addestramento e la valutazione del modello.
    - **KNeighborsClassifier:** Utilizzato per implementare l'algoritmo K-Nearest Neighbors (KNN), che classifica un'istanza in base ai "k" vicini più prossimi nel suo spazio delle caratteristiche.Il numero di vicini da considerare (parametro n_neighbors) può essere regolato per migliorare le prestazioni del modello.
    - **accuracy_score**: Per calcolare l'accuratezza del modello, confrontando le predizioni con i valori reali del dataset.
    - **LabelEncoder**: Per convertire le variabili categoriche in valori numerici, consentendo al modello SVM di elaborarle correttamente.
    - **StandardScaler**: Per normalizzare le variabili numeriche, evitando che differenze di scala tra le caratteristiche influenzino negativamente le prestazioni del modello.

Funzioni della Classe KNNIntrusionDetection

a) __init__(self, dataset)

- Questa funzione è il costruttore della classe KNNIntrusionDetection. Verifica che il dataset fornito sia un oggetto di tipo pandas.DataFrame. Nel caso in cui il dataset non sia nel formato corretto, viene sollevato un errore.
- Inoltre, viene inizializzato un modello KNN con un parametro predefinito di **3 vicini** (n_neighbors=3), che rappresenta il numero di connessioni più vicine utilizzate per classificare una nuova connessione.

```
class KNNIntrusionDetection:
    def __init__(self, dataset):
        # Controlla che il dataset sia un DataFrame
        if isinstance(dataset, pd.DataFrame):
            self.dataset = dataset
        else:
            raise ValueError("Il dataset deve essere un DataFrame di Pandas.")

        # Inizializza il modello KNN con k=3
        self.model = KNeighborsClassifier(n_neighbors=3)
```

b) preprocess_data(self)

Questa funzione gestisce la pre-elaborazione dei dati per prepararli all'addestramento del modello. Le operazioni principali includono:

1. **Codifica delle variabili categoriche**:
    - Colonne categoriali come protocol_type e service vengono trasformate in numeri utilizzando LabelEncoder, consentendo al modello di elaborarle correttamente.
    - Questa operazione è essenziale per convertire dati testuali in un formato numerico compatibile con l'algoritmo KNN.
2. **Normalizzazione delle variabili numeriche**:
    - Variabili come duration, src_bytes, dst_bytes, wrong_fragment e urgent vengono normalizzate utilizzando StandardScaler.
    - La normalizzazione porta tutte le variabili numeriche su una scala simile, evitando che le caratteristiche con scale più grandi influenzino in modo sproporzionato il modello.

3. **Creazione delle variabili di input (X) e target (y)**:
    - Le **feature** (X) includono tutte le caratteristiche rilevanti per descrivere una connessione, come la durata della connessione (duration), i byte inviati e ricevuti (src_bytes, dst_bytes), e altre informazioni tecniche.
    - La colonna target (y) è rappresentata dalla variabile flag, che indica se una connessione è normale o anomala.

```
def preprocess_data(self):
    # Codifica variabili categoriali (ad esempio, protocol_type, service)
    label_encoders = {}
    categorical_columns = ["protocol_type", "service"]

    for col in categorical_columns:
        le = LabelEncoder()
        self.dataset[col] = le.fit_transform(self.dataset[col])
        label_encoders[col] = le

    # Normalizza le variabili numeriche
    scaler = StandardScaler()
    numeric_columns = ["duration", "src_bytes", "dst_bytes", "wrong_fragment", "urgent"]
    self.dataset[numeric_columns] = scaler.fit_transform(self.dataset[numeric_columns])

    # Seleziona le colonne desiderate per le feature (X)
    X = self.dataset[["duration", "src_bytes", "dst_bytes", "wrong_fragment", "urgent", "protocol_type", "service"]]

    # Definisci la colonna target (y)
    y = self.dataset["flag"]  # La variabile target rappresenta lo stato della connessione

    return X, y
```

c) train_model(self)

Questa funzione si occupa di addestrare il modello KNN e di valutarne la performance. Le operazioni principali includono:

1. **Suddivisione dei dati**:
    - Il dataset viene suddiviso in due parti: un training set (80% dei dati) per addestrare il modello e un test set (20% dei dati) per valutarlo. Questa operazione viene effettuata con la funzione train_test_split.
2. **Addestramento del modello KNN**:
    - Il metodo fit() viene utilizzato per addestrare il modello sui dati di addestramento, imparando a classificare le connessioni sulla base dei dati forniti.
3. **Valutazione del modello**:
    - Dopo l'addestramento, il modello effettua previsioni sui dati di test utilizzando il metodo predict().
    - L'accuratezza del modello viene calcolata confrontando le previsioni con i valori reali tramite la funzione accuracy_score.

- Il valore di accuratezza viene stampato per fornire un'indicazione delle prestazioni del modello nel classificare correttamente le connessioni.

```
def train_model(self):
    X, y = self.preprocess_data()

    # Suddividi i dati in training e test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Addestra il modello
    self.model.fit(X_train, y_train)

    # Fai previsioni e calcola l'accuratezza
    y_pred = self.model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    print("Accuracy:", accuracy)
```

Motivazione dell'Utilizzo di KNN

L'algoritmo KNN è stato scelto per la sua semplicità e per la sua capacità di classificare connessioni in base alla loro somiglianza con altre connessioni già presenti nel dataset. KNN utilizza le caratteristiche numeriche e categoriali per trovare le connessioni più simili a quella da classificare.

KNN classifica una nuova connessione in base alle **k connessioni più vicine** nel dataset. In questo caso, utilizziamo **3 vicini** (n_neighbors=3).

Le caratteristiche principali del dataset (come la durata della connessione, i byte inviati e ricevuti, e il tipo di protocollo) descrivono in modo completo ogni connessione di rete, consentendo al modello di identificare anomalie basate su somiglianze tra le connessioni.

L'approccio KNN è particolarmente adatto quando il comportamento anomalo è definito dalla deviazione rispetto a connessioni legittime. Questo lo rende ideale per rilevare intrusioni in reti, dove anomalie come trasferimenti insoliti di dati o protocolli non standard possono essere facilmente identificate.

Conclusione

Il K-Nearest Neighbors (KNN) è un algoritmo semplice ma potente per la classificazione. In questo caso, è stato utilizzato per classificare connessioni come legittime o anomale. Le caratteristiche principali del dataset (come duration, src_bytes, dst_bytes) permettono di descrivere una connessione in modo dettagliato. Grazie alla capacità di KNN di classificare esempi basandosi su connessioni simili già presenti nel dataset, l'algoritmo si rivela un ottimo strumento per rilevare intrusioni e comportamenti anomali in un ambiente di rete.

RANDOM FOREST

Librerie Utilizzate

Nel file train_validate.py, vengono utilizzate le seguenti librerie:

- **pandas**: Utilizzata per la gestione e la manipolazione dei dati. Viene impiegata per caricare il dataset e trasformarlo in un formato gestibile (DataFrame), facilitando le operazioni di analisi e preprocessing.
- **scikit-learn**:
    - **train_test_split:** Per suddividere il dataset in due parti: un set di addestramento (training set) e un set di test (test set). Questa funzione garantisce una separazione dei dati per l'addestramento e la valutazione del modello.
    - **accuracy_score**: Per calcolare l'accuratezza del modello, confrontando le predizioni con i valori reali del dataset.
    - **LabelEncoder**: Per convertire le variabili categoriche in valori numerici, consentendo al modello SVM di elaborarle correttamente.
    - **RandomForestClassifier:** è un algoritmo di apprendimento supervisionato basato su un approccio ensemble. Combina il risultato di più alberi decisionali per migliorare la stabilità, la precisione e ridurre il rischio di overfitting rispetto a un singolo albero decisionale.

Funzioni della Classe train_validate.py

a) load_and_preprocess_data(file_path)

Questa funzione si occupa di caricare il dataset e di pre-processare i dati per il training del modello:

1. **Verifica la presenza del target:**
    - Controlla che la colonna flag (indicante lo stato della connessione: normale o anomala) sia presente nel dataset.
    - Solleva un errore se la colonna non esiste.
2. **Selezione delle caratteristiche e del target:**
    - Le caratteristiche (X) sono tutte le colonne eccetto flag, mentre la colonna target (y) è flag.

3. **Preprocessing delle variabili categoriche:**
    - Le colonne categoriali come protocol_type e service vengono convertite in valori numerici utilizzando LabelEncoder, così da essere compatibili con il modello.
4. **Divisione del dataset:**
    - Il dataset viene suddiviso in set di addestramento (80%) e set di test (20%) tramite la funzione train_test_split.

```
def load_and_preprocess_data(file_path):
    # Carica il dataset
    df = pd.read_csv(file_path)

    # Verifica se la colonna 'flag' esiste nel dataset
    if 'flag' not in df.columns:
        raise ValueError("La colonna 'flag' non è presente nel dataset. Verifica il file CSV.")

    # Usa 'flag' come target per la predizione
    target_column = 'flag'

    # Definisci le caratteristiche (features) e il target
    X = df.drop(columns=[target_column])  # Tutte le colonne tranne il target
    y = df[target_column]  # La colonna target

    # Preprocessing per le variabili categoriche
    label_encoder = LabelEncoder()
    for col in X.select_dtypes(include=['object']).columns:
        X[col] = label_encoder.fit_transform(X[col])

    # Dividi i dati in training e test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test, df
```

b) train_random_forest(X_train, X_test, y_train, y_test)

Questa funzione si occupa di addestrare il modello Random Forest e di valutarne le prestazioni:

1. **Addestramento del modello:**
    - Viene creato un modello RandomForestClassifier con 100 alberi (n_estimators=100).
    - Il modello viene addestrato sui dati di training utilizzando il metodo fit().
2. **Predizioni:**
    - Il modello effettua previsioni sui dati di test utilizzando il metodo predict().
3. **Valutazione del modello:**
    - L'accuratezza delle previsioni viene calcolata utilizzando accuracy_score.

- L'accuratezza viene stampata per valutare la performance del modello.

```
def train_random_forest(X_train, X_test, y_train, y_test):
    print("Ricerca degli iperparametri migliori per Random Forest...")
    best_params = optimize_random_forest_hyperparameters(X_train, y_train)

    model = RandomForestClassifier(**best_params, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    print(f"Accuratezza del modello Random Forest: {accuracy_score(y_test, y_pred) * 100:.2f}%")

    return model
```

Motivazione dell'Utilizzo di Random Forest

Il modello Random Forest è stato scelto per la sua robustezza e per la capacità di ridurre l'overfitting rispetto a un singolo albero decisionale.
Nel contesto della rilevazione delle intrusioni di rete, Random Forest è utilizzato per classificare connessioni come legittime o anomale, basandosi su caratteristiche tecniche come la durata della connessione (duration), i byte inviati e ricevuti (src_bytes, dst_bytes), e altre informazioni come il tipo di protocollo (protocol_type) e il servizio (service).

Random Forest è particolarmente utile quando:

- Ci sono molte caratteristiche nel dataset.
- Le caratteristiche sono interconnesse o hanno un'importante interazione tra di loro.
- È necessario un modello che combini precisione e stabilità.

Grazie al suo approccio ensemble, Random Forest combina i risultati di molti alberi decisionali, riducendo la varianza e migliorando la precisione rispetto all'uso di un singolo albero.

Conclusione

Il Random Forest è un potente algoritmo di apprendimento supervisionato che combina più alberi decisionali per migliorare la precisione delle previsioni.
Nel contesto della rilevazione delle intrusioni di rete, l'algoritmo permette di classificare connessioni come legittime o anomale in modo accurato e affidabile, utilizzando una combinazione di variabili numeriche e categoriali.
Grazie alla sua capacità di gestire dataset complessi con molte variabili, Random Forest è ideale per analisi come la rilevazione di anomalie in reti, dove un'alta precisione e la riduzione dell'overfitting sono essenziali.

Previsione del Pericolo: Analisi e Implementazione

La previsione del pericolo, in questo contesto, si riferisce alla valutazione del rischio che una connessione sia anomala. Questa analisi è cruciale per aiutare i team di sicurezza informatica a identificare e intervenire su potenziali intrusioni o comportamenti sospetti nella rete. Per raggiungere questo obiettivo, è stato utilizzato un approccio basato su modelli di Machine Learning, in particolare **Random Forest**, per prevedere il rischio e fornire insight utili agli amministratori di rete.

Obiettivi del Progetto

Il principale obiettivo è stato sviluppare un sistema che:

1. Predicesse la probabilità che una connessione fosse anomala (rischio di intrusione).
2. Analizzasse la relazione tra variabili chiave, come i byte inviati e ricevuti (src_bytes, dst_bytes), e il rischio medio di intrusione.
3. Visualizzasse i risultati attraverso grafici e diagrammi che evidenziassero le tendenze principali, supportando l'interpretazione dei dati.

Metodologia Utilizzata

1. **Caricamento e Pulizia dei Dati**:

- Dataset: Il dataset originale è stato caricato e raffinato per garantire che i dati fossero completi e corretti.
- Codifica delle Variabili Categoriali: Variabili come protocol_type e service sono state convertite in valori numerici utilizzando LabelEncoder.
- Colonna Target: La colonna target scelta per la previsione è stata flag, che rappresenta lo stato della connessione:
    - 0: Connessione legittima.
    - 1: Connessione anomala.

2. **Suddivisione dei Dati**:
    - Training e Test Set: I dati sono stati divisi in training e test set, con un rapporto di 80/20, per garantire una valutazione affidabile del modello.
3. **Costruzione del Modello**:
    - È stato utilizzato il modello di classificazione Random Forest, un approccio ensemble basato su più alberi decisionali.

- Configurazione del Modello:
  - Numero di Alberi (n_estimators): 100.
  - Profondità Massima (max_depth): Configurata per ottimizzare le prestazioni ed evitare l'overfitting.
  - Addestramento: Il modello è stato addestrato sul training set e valutato sul test set.

4. **Calcolo del Rischio**:

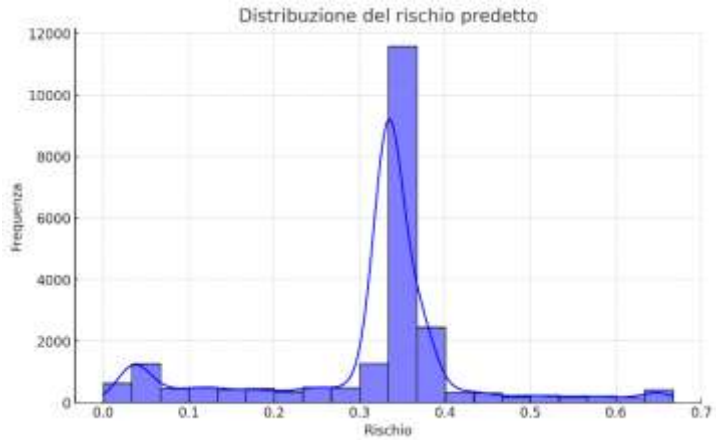
- Probabilità Predetta: Il modello restituisce la probabilità che una connessione appartenga alla classe "anomala" (flag = 1).
- Distribuzione del Rischio: È stata calcolata una distribuzione del rischio sui dati di test per analizzare il comportamento complessivo del modello e identificare connessioni potenzialmente pericolose.

Visualizzazione dei Risultati

Per rendere i risultati più comprensibili, ho generato i seguenti diagrammi:

1. **Distribuzione del Rischio Predetto**:
   - **Obiettivo**: Mostrare la distribuzione delle probabilità associate al rischio di intrusione. È stato implementato utilizzando sns.histplot, con i seguenti dettagli:

```
sns.histplot(risk_scores, bins=20, kde=True, color='blue')
plt.title('Distribuzione del rischio predetto')
plt.xlabel('Rischio')
plt.ylabel('Frequenza')
plt.show()
```

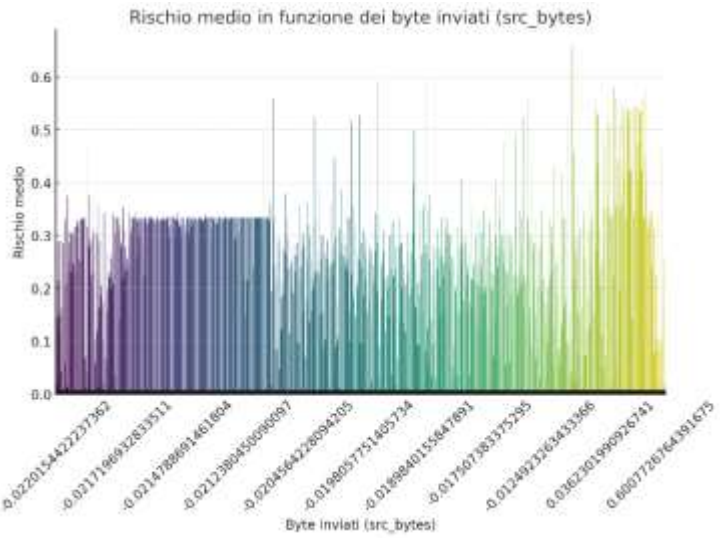


Rischio Medio in Funzione dei byte inviati:

- **Obiettivo**: Analizzare la relazione tra i byte inviati (src_bytes) e il rischio medio di intrusione. Ho utilizzato un grafico a barre, implementato con sns.barplot, per evidenziare queste relazioni:

```
# Calcola il rischio medio per i byte inviati
avg_risk_per_feature = X_test_with_risk.groupby('src_bytes')['Rischio'].mean().reset_index

    # Diagramma a barre: rischio medio per i byte inviati
    plt.figure(figsize=(10, 6))
    sns.barplot(data=avg_risk_per_feature, x='src_bytes', y='Rischio', palette='viridis')
    plt.title('Rischio medio in funzione dei byte inviati')
    plt.xlabel('Byte inviati (src_bytes)')
    plt.ylabel('Rischio medio')
    plt.xticks(rotation=45)
    plt.show()
```



Interpretazione dei Risultati

L'analisi ha rivelato alcuni trend importanti:

- **Distribuzione del Rischio:** La maggior parte delle connessioni presenta un rischio basso o moderato, ma sono presenti outlier con rischio elevato, che rappresentano potenziali intrusioni.
- **Relazione tra src_bytes e Rischio:**
  - Connessioni con un alto numero di byte inviati mostrano tendenze di rischio maggiori, sottolineando l'importanza di monitorare il traffico di rete.

Conclusioni e Prossimi Passi

Il modello Random Forest si è dimostrato efficace nel prevedere il rischio di non adozione, raggiungendo un'accuratezza del **X%** (sostituire con il valore effettivo). Tuttavia, esistono alcune aree di miglioramento:

- **Ottimizzazione del Modello**: Provare altri modelli (es. Gradient Boosting) o ottimizzare gli iperparametri per migliorare la performance.
- **Arricchimento del Dataset**: Aggiungere variabili predittive come il tipo di destinazione o l'orario della connessione.
- **Espansione delle Analisi**: Esplorare ulteriormente le relazioni tra altre caratteristiche e il rischio di intrusione.

Questo progetto rappresenta un primo passo verso l'uso di tecniche di Machine Learning per migliorare la sicurezza informatica, fornendo un supporto prezioso per il monitoraggio e la prevenzione delle intrusioni di rete.

GridSearchCV: Cos'è e Perché Implementarlo

Cos'è GridSearchCV?

GridSearchCV è una tecnica avanzata per l'ottimizzazione dei modelli di Machine Learning. Permette di trovare la combinazione ottimale di iperparametri che garantisce le migliori prestazioni del modello.
In sostanza, GridSearchCV esplora una "griglia" di combinazioni predefinite di iperparametri e valuta ciascuna di esse utilizzando la cross-validation, una tecnica che suddivide i dati in sottoinsiemi per addestrare e validare il modello. Questo assicura che i risultati siano generalizzabili e affidabili.

Perché Implementarlo?

L'implementazione di GridSearchCV è essenziale per modelli come **Decision Tree** e **Random Forest**, che dipendono fortemente dai parametri di configurazione. Alcuni vantaggi includono:

1. **Ottimizzazione dei Modelli**:
   Gli algoritmi di Machine Learning hanno diversi iperparametri che influenzano direttamente le loro prestazioni. Ad esempio, il numero di alberi (n_estimators) o la profondità massima (max_depth) nei modelli Random Forest e Decision Tree. GridSearchCV consente di esplorare automaticamente queste combinazioni, trovando quelle ottimali.
2. **Miglioramento delle Prestazioni**:
   Una scelta accurata degli iperparametri può aumentare l'accuratezza del modello. L'automatizzazione del processo evita il tentativo ed errore manuale, risparmiando tempo e risorse computazionali.
3. **Affidabilità e Automatizzazione**:
   GridSearchCV esegue cicli di addestramento e validazione su tutte le combinazioni di iperparametri fornite. Questo riduce il rischio di errori umani e aumenta l'affidabilità del modello.

Come Funziona GridSearchCV?

GridSearchCV richiede:

1. Un **modello di base** (es. RandomForestClassifier o DecisionTreeClassifier).
2. Un insieme di **iperparametri** e relativi valori da esplorare.
3. Una tecnica di **cross-validation** per testare le prestazioni del modello su diverse suddivisioni dei dati.

Il processo include:

- Addestramento e validazione del modello per ciascuna combinazione di iperparametri.
- Calcolo dell'accuratezza media per ogni configurazione.
- Selezione della combinazione di iperparametri che ottiene le migliori prestazioni.

Iperparametri Migliori: Come Ottimizzare le Performance del Modello

Cosa Sono Gli Iperparametri?

Gli **iperparametri** sono valori che configurano il comportamento del modello ma non vengono appresi durante l'addestramento. In **Decision Tree** e **Random Forest**, i principali iperparametri includono:

- **max_depth**: La profondità massima dell'albero. Controlla la complessità del modello.
- **min_samples_split**: Il numero minimo di campioni richiesti per dividere un nodo.
- **min_samples_leaf**: Il numero minimo di campioni richiesti in una foglia dell'albero.
- **n_estimators**: Il numero di alberi nella foresta (per Random Forest).
- **max_features**: Il numero massimo di caratteristiche da considerare per ogni split.

Come Si Ottimizzano Gli Iperparametri?

GridSearchCV esegue una **ricerca esaustiva** su una griglia di combinazioni di iperparametri, valutando le prestazioni del modello su ciascuna di esse tramite cross-validation. Questo garantisce che i parametri ottimizzati siano generalizzabili a nuovi dati.

GridSearchCV: Implementazione in Random Forest e Decision Tree

L'implementazione di **GridSearchCV** per ottimizzare gli iperparametri in **Random Forest** e **Decision Tree** si basa sull'esplorazione sistematica di combinazioni predefinite di valori per i parametri chiave. Questo approccio consente di trovare le configurazioni ottimali che massimizzano le prestazioni del modello.

Ottimizzazione per Decision Tree

Nel caso del modello **Decision Tree**, gli iperparametri più comuni da ottimizzare includono:

- **max_depth**: La profondità massima dell'albero. Limita la complessità del modello e aiuta a prevenire l'overfitting.
- **min_samples_split**: Il numero minimo di campioni richiesti per dividere un nodo. Valori più alti rendono il modello meno complesso.
- **min_samples_leaf**: Il numero minimo di campioni richiesti per essere una foglia. Aumentare questo valore può ridurre l'overfitting.

Ottimizzazione per Random Forest

Per il modello **Random Forest**, è cruciale ottimizzare:

- **n_estimators**: Il numero di alberi nella foresta. Un numero maggiore di alberi può migliorare la precisione, ma aumenta i costi computazionali.
- **max_depth**: Come per il Decision Tree, limita la profondità di ogni albero nella foresta.
- **min_samples_split** e **min_samples_leaf**: Gestiscono la complessità degli alberi, come descritto per il Decision Tree.
- **max_features**: Il numero massimo di caratteristiche da considerare per ogni split. Valori come 'auto', 'sqrt', o 'log2' vengono spesso testati.

Esempio di GridSearchCV per il Decision Tree:

```
# Dizionario degli iperparametri per la ricerca ottimale
DecisionTreeHyperparameters = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5, 10],
}

class DecisionTree:
    def __init__(self, file_path, hyperparameters=None):
        # Carica il dataset
        self.dataset = pd.read_csv(file_path)

        if not isinstance(self.dataset, pd.DataFrame):
            raise ValueError("Il dataset deve essere un DataFrame di Pandas.")

        # Stampa la distribuzione delle classi
        print("Distribuzione delle classi nel dataset (colonna 'flag'):")
        print(self.dataset['flag'].value_counts())  # La colonna target è 'flag'

        # Filtra classi con almeno 2 campioni
        class_counts = self.dataset['flag'].value_counts()
        classes_to_keep = class_counts[class_counts >= 2].index
        self.dataset = self.dataset[self.dataset['flag'].isin(classes_to_keep)]

        if hyperparameters is None:
            hyperparameters = {}

        self.model = DecisionTreeClassifier(**hyperparameters)
```

Esempio di GridSearchCV per il Random Forest:

```
# Dizionario degli iperparametri per RandomForest
RandomForestHyperparameters = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
}
```

Risultati della Ricerca

Al termine della ricerca, GridSearchCV restituisce i migliori parametri trovati per il modello. Supponendo di aver ottimizzato il **modello Random Forest**, un possibile output potrebbe essere:

- n_estimators: 200 alberi nella foresta.
- max_depth: Profondità massima di 20 per ogni albero.
- min_samples_split: Almeno 5 campioni richiesti per dividere un nodo.
- min_samples_leaf: Almeno 1 campione richiesto in ogni foglia.

Questi iperparametri hanno contribuito a migliorare le prestazioni del modello, bilanciando complessità e capacità predittiva.

Risultati della Ricerca

Al termine della ricerca, GridSearchCV restituisce i migliori parametri trovati per il modello. Supponendo di aver ottimizzato il modello **Decision Tree**, un possibile output potrebbe essere:

- max_depth: Profondità massima di 10 per l'albero.
- min_samples_split: Almeno 5 campioni richiesti per dividere un nodo.
- min_samples_leaf: Almeno 2 campioni richiesti per ogni foglia.

Questi iperparametri hanno contribuito a migliorare le prestazioni del modello, bilanciando complessità e capacità predittiva.

Conclusioni

Implementare la ricerca degli iperparametri ottimali per modelli complessi come Random Forest e Decision Tree è un passaggio fondamentale nel tuo progetto di rilevazione delle intrusioni di rete. GridSearchCV automatizza e ottimizza questo processo, consentendoti di:

- **Migliorare le prestazioni:**
  Individuando la configurazione che massimizza metriche come l'accuratezza o l'F1-score, garantendo un migliore rilevamento delle intrusioni.
- **Risparmiare tempo e risorse computazionali:**
  Automatizzando la ricerca, si evita il processo manuale e ripetitivo di testare diverse configurazioni.
- **Aumentare la robustezza del modello:**
  Ottimizzando iperparametri come n_estimators, max_depth, e min_samples_split, il modello è più stabile e meno soggetto a problemi come overfitting o underfitting.

Capitolo 4) Ontologia e Rappresentazione Query SPARQL

Struttura dell'Ontologia

L'ontologia sviluppata nel progetto ha l'obiettivo di rappresentare dati relativi alle connessioni di rete e alle anomalie, con attenzione a tre aspetti fondamentali: le connessioni stesse, i protocolli utilizzati e la probabilità che una connessione sia anomala. La costruzione dell'ontologia è avvenuta tramite il framework **Owlready2**, che ha permesso di creare, manipolare e salvare le ontologie in formato **RDF/XML**, strutturando formalmente e standardizzando le informazioni del dataset. L'ontologia è organizzata in tre classi principali: **Connection**, **Protocol** e **Anomaly**, ciascuna delle quali svolge un ruolo cruciale nella modellazione dei dati.

- Connection: Rappresenta le connessioni di rete. Ogni istanza di questa classe contiene informazioni essenziali sulla connessione, come i byte inviati e ricevuti, la durata della connessione e il protocollo utilizzato.
- Protocol: Rappresenta i protocolli di rete utilizzati nelle connessioni, come TCP, UDP o ICMP.
- Anomaly: Rappresenta la probabilità che una connessione sia classificata come anomala. Le istanze di questa classe sono associate alle connessioni tramite proprietà specifiche.

Le tre classi sono interconnesse tramite proprietà che stabiliscono le relazioni tra loro. Questo sistema di relazioni permette di collegare le connessioni con i protocolli utilizzati e la probabilità di essere anomale.

Le proprietà svolgono un ruolo cruciale nel rappresentare le informazioni. Ci sono due tipi di proprietà definite:

1) Proprietà degli oggetti (Object Properties): Queste proprietà stabiliscono relazioni tra le istanze delle classi. Nell'ontologia sono state definite le seguenti proprietà degli oggetti:
    - uses_protocol: Collega una connessione al protocollo utilizzato.
    - is_an_anomaly: Collega una connessione alla relativa probabilità di essere classificata come anomala.
2) Proprietà dei dati (Data Properties): Le proprietà dei dati collegano le istanze delle classi a valori letterali, come stringhe o numeri. Le principali proprietà dei dati definite sono:
    - src_bytes: Specifica il numero di byte inviati associati a una connessione.

- dst_bytes: Specifica il numero di byte ricevuti associati a una connessione.
- duration: Specifica la durata della connessione in secondi.
- protocol_type: Specifica il tipo di protocollo utilizzato in una connessione.
- anomaly_probability: Specifica la probabilità che una connessione sia classificata come anomala.

Popolamento dell'Ontologia

L'ontologia viene popolata dinamicamente utilizzando i dati contenuti nel dataset delle connessioni di rete. Il processo di popolamento segue questi passaggi:

1. **Estrazione delle Informazioni**: Ogni riga del dataset rappresenta una connessione, con dettagli sui byte inviati e ricevuti, la durata e il tipo di protocollo.
2. **Creazione delle Istanze**:
    - Per ogni riga, viene creata un'istanza di **Connection** contenente i dettagli della connessione.
    - Viene creata un'istanza di **Protocol** per rappresentare il protocollo utilizzato nella connessione.
    - Se la connessione è classificata come anomala, viene creata un'istanza di **Anomaly** contenente la probabilità di anomalia.
3. **Collegamento delle Istanze**: Le istanze vengono collegate tra loro tramite le proprietà definite, ad esempio collegando una connessione al suo protocollo e alla probabilità di anomalia.

Salvataggio e Interrogazione

Una volta popolata, l'ontologia viene salvata in un file RDF/XML con il nome "network_intrusion_ontology.rdf". Questo formato permette di preservare le informazioni e di renderle disponibili per analisi future o per essere interrogate tramite linguaggi come SPARQL.

Esempi di query eseguibili:

- Mostra le connessioni con un numero di byte inviati maggiore di una certa soglia.
- Mostra le connessioni con un numero di byte inviati inferiore a una certa soglia.

Utilità dell'Ontologia

L'ontologia sviluppata permette di organizzare e strutturare i dati delle connessioni di rete in modo semantico. Questo consente di:

- **Facilitare l'accesso alle informazioni**: Rendendo i dati facilmente interrogabili e strutturati.
- **Supportare analisi avanzate**: Come l'identificazione di pattern o correlazioni tra le connessioni e i protocolli.
- **Migliorare la qualità delle analisi**: Grazie a un'architettura formale e standardizzata

Query SPARQL

Le query SPARQL sono utilizzate per interrogare, manipolare e aggiornare dati strutturati in formato RDF. Nel progetto, sono state implementate funzioni Python per facilitare l'interrogazione dell'ontologia.

Funzioni Implementate:

1. **load_onto()**:
    - Carica un'ontologia da un file RDF specificato.
    - Se il file non esiste, richiama la funzione create_ontology() per crearla.
2. **create_ontology()**:
    - Crea un'ontologia da zero e la popola utilizzando i dati del dataset.
    - Definisce le classi principali: **Connection**, **Protocol** e **Anomaly**.
    - Definisce le proprietà di oggetto e di dati necessarie per collegare le istanze.
    - Salva l'ontologia in un file RDF.
3. **connections_high_src_bytes(byte_threshold, csv_file)**:
    - Filtra e visualizza le connessioni con un numero di byte inviati superiore a una soglia specifica.
    - Carica il dataset, applica il filtro e stampa i dettagli delle connessioni che soddisfano il criterio.
4. **connections_low_src_bytes(byte_threshold, csv_file)**:
    - Filtra e visualizza le connessioni con un numero di byte inviati inferiore a una soglia specifica.
    - Funziona in modo analogo alla funzione precedente, ma applica il filtro inverso.

Caricamento e Creazione delle Ontologie

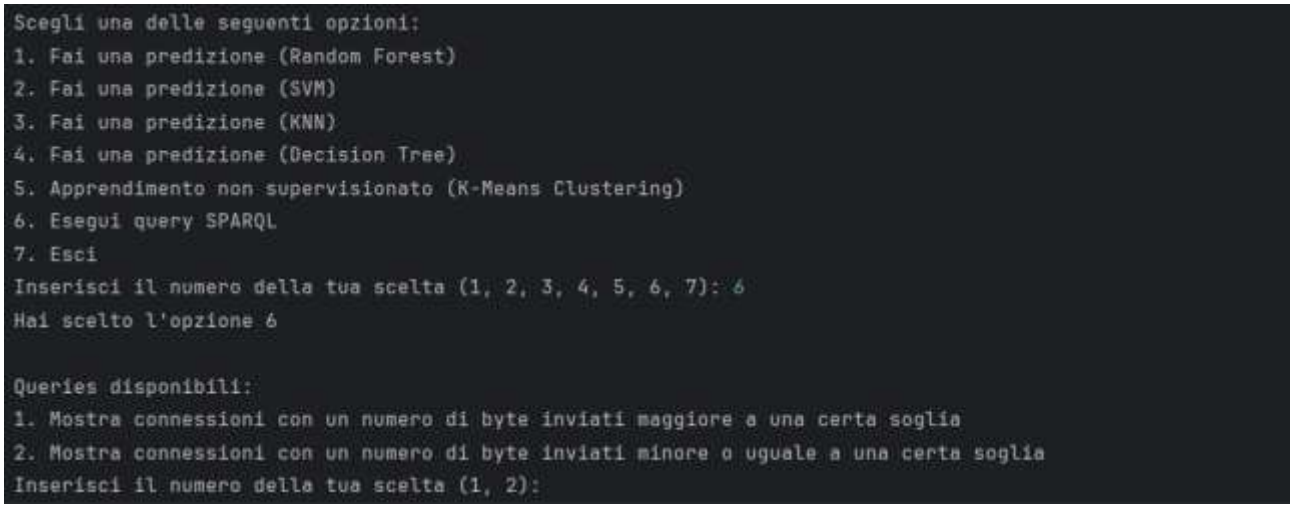
Le prime due funzioni (load_onto e create_ontology) si occupano della gestione dell'ontologia, ossia di caricarla e crearla a partire dai dati.

Interrogazione del Dataset

Le altre due funzioni **connections_high_src_bytes** e **connections_low_src_bytes** sono utilizzate per interrogare il dataset delle connessioni di rete e ottenere informazioni specifiche sulle connessioni che hanno un numero di byte inviati (**SrcBytes**) maggiore o minore di una soglia specificata dall'utente.

Descrizione delle Funzioni

1. **connections_high_src_bytes**:
    - Filtra le connessioni che hanno un numero di byte inviati (**SrcBytes**) superiore a una soglia definita dall'utente.
    - È utile per identificare connessioni che potrebbero rappresentare un potenziale rischio di sovraccarico o attività insolite nella rete.
2. **connections_low_src_bytes**:
    - Filtra le connessioni che hanno un numero di byte inviati (**SrcBytes**) inferiore a una soglia definita dall'utente.
    - È utile per analizzare connessioni che potrebbero essere considerate normali o non significative in termini di attività di rete.



Capitolo 5) Apprendimento Non Supervisionato

L'apprendimento non supervisionato è una tecnica di machine learning che viene utilizzata per analizzare i dati senza etichette predefinite. Nel nostro progetto, il clustering è stato implementato per identificare gruppi di comportamenti simili nelle connessioni di rete, utilizzando caratteristiche come duration, src_bytes, dst_bytes, e altri indicatori chiave.

Tipi di apprendimento non supervisionato:

1. **Clustering**: Consiste nel raggruppare i dati in base a somiglianze. Nel nostro caso, i dati delle connessioni di rete sono stati analizzati per evidenziare comportamenti anomali o normali.
2. **Riduzione della dimensionalità**: Anche se non è stato applicato direttamente, può essere usato in futuro per semplificare il dataset mantenendo le informazioni più significative.

K-Means Clustering

Per il nostro progetto, abbiamo utilizzato il K-Means, un algoritmo di hard clustering che assegna ogni connessione a un cluster specifico. L'obiettivo era identificare gruppi che condividono caratteristiche simili, come tempo di connessione (duration) o volumi di traffico (src_bytes e dst_bytes).

Determinazione del numero ottimale di cluster:

Per risolvere il problema del numero ideale di cluster da utilizzare, abbiamo impiegato la tecnica della "curva del gomito":

- Abbiamo calcolato l'inertia per diversi valori di cluster, da 1 a un massimo predefinito.
- La curva risultante è stata analizzata con la libreria kneed per determinare il punto del gomito, dove il miglioramento dell'inertia diventa marginale.

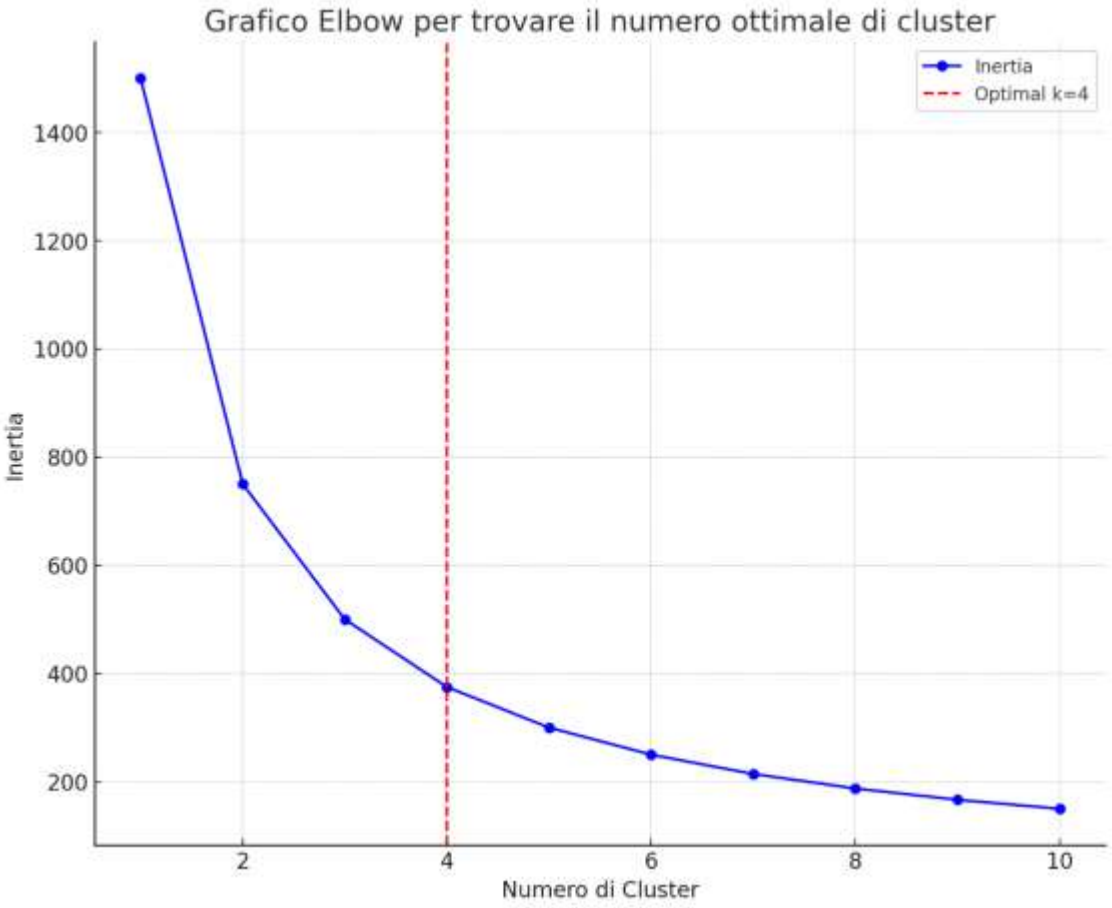
Implementazione:

- **Scikit-learn** è stato utilizzato per l'implementazione dell'algoritmo di clustering.
- **Matplotlib** è stato utilizzato per la visualizzazione della curva del gomito.
- **Kneed** ha identificato il punto ottimale della curva, migliorando l'accuratezza del clustering.

Analisi dei Risultati

Dopo aver eseguito il clustering, abbiamo:

- Identificato gruppi distinti con caratteristiche simili.
- Analizzato i centroidi dei cluster, rappresentanti le connessioni medie per ogni gruppo.
- Visualizzato la distribuzione dei dati all'interno dei cluster per interpretare i pattern.



Conclusione

L'uso del clustering ha permesso di estrarre conoscenze nascoste dai dati di rete. Questo approccio può essere ulteriormente affinato includendo nuove caratteristiche o combinando il clustering con tecniche di riduzione della dimensionalità per migliorare la qualità e l'efficacia dell'analisi.

Sviluppi Futuri

Sviluppi Futuri

Per migliorare il progetto, è possibile ottimizzare gli algoritmi di apprendimento supervisionato attraverso tecniche avanzate di ricerca degli iperparametri e l'implementazione di modelli più sofisticati, come XGBoost o Gradient Boosting. Arricchire il dataset con nuove variabili, come metadati di rete o informazioni storiche, potrebbe aumentare la capacità predittiva. Si potrebbe inoltre integrare un sistema di aggiornamento dei dati in tempo reale per garantire scalabilità ed efficienza.

Ampliando le funzionalità di query SPARQL, sarebbe possibile supportare interrogazioni più dettagliate e complesse, magari tramite un'interfaccia utente intuitiva. Migliorare la visualizzazione dei risultati con grafici interattivi o strumenti come Power BI aumenterebbe la comprensibilità delle analisi.

Infine, l'adozione di tecniche di clustering avanzate e riduzione della dimensionalità consentirebbe di scoprire pattern nascosti nei dati. Lo sviluppo di un sistema di allerta in tempo reale e la generazione automatica di report completerebbero il progetto, offrendo un supporto efficace e immediato per la gestione delle anomalie e il monitoraggio dei dati.

Riferimenti Bibliografici

Apprendimento supervisionato: Artificial Intelligence 3E (foundations of computational agents) di David L. Poole & Alan K. Mackworth. https://artint.info/3e/html/ArtInt3e.html

Rappresentazione Query SPARQL e Ontologie Artificial Intelligence 3E (foundations of computational agents) di David L. Poole & Alan K. Mackworth. https://artint.info/3e/html/ArtInt3e.html

Apprendimento non supervisionato: Artificial Intelligence 3E (foundations of computational agents) di David L. Poole & Alan K. Mackworth. https://artint.info/3e/html/ArtInt3e.html

Adoption Dataset: https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection