OverseerAI

https://github.com/BiBooBap/OverseerAI

Laurea di Informatica L-31 all'Università di Salerno

Corso di Fondamenti di Intelligenza Artificiale

Gennaio 2025

Created by: Antonio Maiorano

Supervised by: Prof. Fabio Palomba

Indice

CHAPTER 1

Definizione del problema

1.1 Introduzione

YouTube è una delle piattaforme di condivisione video più utilizzate al mondo. Tuttavia, con l'aumento dei contenuti, sono cresciuti anche i tentativi di truffa, certe volte mascherati con il metodo "*fare tanto, dando poco*".
Per semplicità di analisi, i contenuti "truffa" sono una generalizzazione di tutti quei contenuti che portano l'utente finale a consumare contenuti che sono descritti in modo fuorviante e/o sbagliato, si includono dunque i contenuti che reindirizzano ad altri contenuti esterni alla piattaforma per lo stesso obiettivo.
Queste problematiche sono ora piu' presenti che mai a causa della rapida crescita della presenza online dei contenuti AI-Made (le cosiddette "Money Farm") e pubblicità sponsorizzate da individui singoli con intenzioni a volte poco chiare, e poco legittime.

1.2 Obiettivi

L'obiettivo principale di questo progetto è quello di evitare il piu' possibile un contatto diretto con questi contenuti-truffa creando un modello di Machine Learning in grado di classificare i titoli dei video.

1.3 Specifica P.E.A.S.

1.3.1 Performance

La misura delle prestazioni del modello:

- Correttezza di classificazione dei titoli.

1.3.2 Environment

Descrizione degli elementi che formano l'ambiente in cui il modello opererà:

- Dataset sintetico di titoli creati con l'aiuto di Large Language Models, i cui elementi saranno descritti di seguito (applicati limiti derivanti dalla piattaforma stessa).

1.3.3 Actuators

Gli attuatori che prenderanno le decisioni nel modello:

- Classificatore che determinerà l'appartenenza di un titolo ad una delle due categorie.

1.3.4 Sensors

I sensori attraverso cui il modello riceve le percezioni su cui opererà:

- La finestra di input.

1.4 Caratteristiche dell'Ambiente

- **Osservabilità:** Parzialmente osservabile.
  - Il modello può osservare ed analizzare solo le caratteristiche del video e non il contenuto del video stesso.
- **Determinismo:** Parzialmente stocastico.
  - Presenza di ambiguità e rumore all'interno delle caratteristiche in analisi.
- **Episodicità:** Episodico.
  - Ogni titolo viene esaminato indipendentemente dagli altri
- **Dinamismo:** Statico.
  - I dati usati dal modello sono costanti, e non subiscono cambiamenti.
- **Tipo di agente:** Singolo agente.
  - Vi è un solo agente ad analizzare i titoli.

1.5 Metodologia

La valutazione della metodologia di approccio al problema in analisi è stata fatta tra i modelli di Ingegneria del Machine Learning **CRISP-DM** e **TDSP**, dove è stato scelto il primo, data la non necessità di suddivisioni e validazioni temporali del progetto presenti nel secondo.

1.5.1 CRISP-DM

Le fasi che verranno percorse sono:

1. Business Understanding.
2. Data Understanding.
3. Data Preparation.
4. Data Modeling.
5. Evaluation.

CHAPTER 2

Business Understanding

2.1 Determinare gli Obiettivi di Business

Il primo passo per la creazione di OverseerAI, è capire cosa esso deve essere, come deve esserlo e come esso deve arrivare ad essere cio' che è stato descritto di esso.
Partendo dalla radice: il problema in analisi è un problema di classificazione, come vedremo più avanti esso è nello specifico di **classificazione binaria**.

2.1.1 Background

Esistono molti sistemi di "Auto-Flagging", o di Moderazione automatica, YouTube stesso ne ha almeno uno!
...Il problema è che i contenuti moderati sono solo quelli degli utenti, ad esempio le pubblicità non sono moderate, per qualche ragione di Business di Google.
Un sistema esistente (e dignitosamente più funzionante dell'esempio appena esplicitato) da cui OverseerAI potrebbe prendere spunto:

- **La moderazione automatica del social media TikTok**

Il suddetto sistema funziona sulla base di un algoritmo (Closed-Source) che modera i contenuti in modo automatico basandosi su similarità con i contenuti di allenamento, che, nel caso di TikTok, possono essere Hashtags riconosciuti come pericolosi (la ricerca sulla piattaforma di tali Hashtags porta zero risultati), oppure delle parole presenti nella descrizione che sono anche essere già conosciute come pericolose.
Con grande probabilità esso utilizza anche un algoritmo per l'analisi dei contenuti.

2.1.2 Obiettivi di Business

Basandoci sulle informazioni raccolte, OverseerAI dovrà analizzare TUTTI i contenuti di YouTube, in particolare i loro titoli. Assieme ad essi si potrebbe analizzare la presenza di link all'interno della descrizione, che sono la maggior parte delle volte usati per truffare e rubare dati alle persone ignare.
Ovviamente quest'ultimo puo' facilmente essere un falso positivo, in quanto ci sono anche canali leciti che usano i link per aiutare le persone a connettere con loro attraverso altri social media.

2.2 Criteri di successo

Gli obiettivi, non per ordine di importanza, di OverseerAI dunque sono:

- **Un corretto riconoscimento dei contenuti intenzionati a truffare**
- **Un corretta classificazione dei contenuti normali come leciti**
- **Evitare di creare bias con chi ha link nei propri contenuti**

Ovviamente, assieme ad essi, c'è anche il criterio di aver sventato delle truffe!

2.3 Limiti

A causa di un recente cambiamento nella piattaforma di Youtube, vi è un limite alla consumazione di contenuti, soprattutto se l'utente non ha eseguito il Login.
Inoltre, anche lo scraping stesso del sito è stato limitato di molto (causa).
Volendo evitare un aumento esponenziale della complessità del progetto, dovuto anche al dover ricercare manualmente le entries da aggiungere al dataset, e volendo rimanere in un'ottica più generale, l'intenzione iniziale di usare dati reali ottenuti dalla piattaforma stessa è stata accantonata, ai dati creati sinteticamente tramite l'uso di LLMs e tramite l'utilizzo di scripts python di potenziamento.

2.4 Tecnologie Usate

Le principali tecnologie usate sono:

- I **LLMs** come tool di supporto;
- **Python** come linguaggio di programmazione;
  - Diverse librerie usate per le funzioni critiche, tra le quali `pandas` per alcune funzioni base eseguite sul Dataset, `sklearn` per le funzioni di Machine Learning, etc...

CHAPTER 3

Data Understanding

3.1 Ottenimento dei Dati

Come descritto in precedenza, a causa di alcuni limiti non ci sarà possibile utilizzare dati reali.
Al loro posto, utilizzeremo dei dati creati sinteticamente tramite l'uso di LLMs.
Dopo l'istruzione, del suddetto, a riguardo del problema in esame, e del contesto ad esso relativo, non vi è granchè bisogno di fare Prompt Engineering: richiedendo semplicemente la generazione delle istanze, e per ogni iterazione essere sicuri di non averne di ripetute per evitare problemi, esso ne genererà continuamente.
Le istanze generate sono istanze di struttura CSV, dove vi sono solo due features:

- Il **titolo**, `title`;
- Una **etichetta, Target Feature del dataset**, `label`, che può essere "scam" o "legit", e determina se un video è truffaldino o meno;
- Una **descrizione**, `description`, che contiene testo e può contenere link al suo interno.

Le istanze sono generate solo con queste tre features causa la tipologia delle features stesse: sono strettamente collegate e devono avere senso nel contesto dell'istanza, dunque non possono essere generate come le altre features, invece indipendenti tra loro.
Inoltre le istanze verranno generate in inglese per semplicità.

3.2 Miglioramento del Dataset

3.2.1 Aggiunta di features

I dati creati sinteticamente mancano di alcune features collegate ai contenuti.
Ovviamo a questo problema creando da zero dei dati realistici, con funzioni randomiche basate su osservazioni soggettive dei casi reali.
Le features riguardante i contenuti che verranno create sono:

- I **mi piace** come `likes`;
- I **non mi piace** come `dislikes`;

- L' **ora di caricamento** come `upload_hour`;
- I **commenti** come `comments`.

Dopo aver creato le features nel dataset, procederemo ad assegnare ad ogni istanza dei valori ad essi riguardanti.

```
import pandas as pd
import random
import numpy as np

def add_features(df):
    df['upload_hour'] = np.random.randint(0, 24, df.shape[0])
    df['likes'] = np.random.randint(0, 100001, df.shape[0])
    df['dislikes'] = np.random.randint(0, 10001, df.shape[0])
    df['comments'] = np.random.randint(0, 10001, df.shape[0])
```

3.2.2 Aggiunta di rumore

Il dataset risultante è vicino ad alla realtà, ma per renderlo ancor più realistico aggiungiamo del rumore nei dati, molto spesso presente in qualsiasi dataset.
Per il 3% delle istanze del dataset, viene introdotto rumore per la feature "titolo", e la tipologia di rumore viene introdotto randomicamente nella scelta di uno dei seguenti:

- Sostituzione di caratteri con caratteri casuali;
- Aggiunta di caratteri speciali;
- Mescolamento dei caratteri originali;
- Testo generico di errore.

Per il 5% delle istanze del dataset, viene introdotto rumore per una delle features tra "upload_hour", "likes", "dislikes" o "comments", immettendo valori vuoti "NaN".
Per il 5% delle istanze del dataset, viene introdotto rumore per le features "likes" e "dislikes" immetendo valori anomali, in questo caso valori assurdi.

```
import pandas as pd
import random
import numpy as np
def add_noise(df):
    def corrupt_title(title):
        if pd.isnull(title):
            return title

        noise_methods = [
            lambda x: ''.join(random.choice(string.ascii_letters + string.digits + string.punctuation) for _ in range(len(x))),
            lambda x: x + ''.join(random.choices(string.punctuation, k=5)),
            lambda x: ''.join(random.sample(x, len(x))),
            lambda x: '###ERROR###'
        ]

        return random.choice(noise_methods)(title)

    noise_indices = df.sample(frac=0.03).index
    df.loc[noise_indices, 'title'] = df.loc[noise_indices, 'title'].apply(corrupt_title)

    for col in ['upload_hour', 'likes', 'comments']:
        df.loc[df.sample(frac=0.05).index, col] = np.nan

    noise_indices = df.sample(frac=0.05).index
    (`len(noise_indices)`)
    df.loc[noise_indices, 'likes'] = np.random.randint(1000000, 10000000, len(noise_indices))
    df.loc[noise_indices, 'dislikes'] = np.random.randint(500000, 1000000, len(noise_indices))

    return df
```

CHAPTER 4

Data Preparation

4.1 Preparazione del dataset

Il dataset in uso presenta tipi diversi di rumore: su tutte le features, ci sono dei valori anomali che potrebbero creare problematiche anche rilevanti durante la creazione e l'utilizzo del modello.
Prima di passare dunque alla suddetta fase, il dataset deve essere pulito.
Inoltre, i valori numerici potrebbero essere problematici in fase di training, in quanto risulta avere valori senza un contesto: un esempio semplice può essere `upload_hour` che è un valore numerico con un contesto specifico, ovvero quello delle ore in una giornata, e non ha dunque senso unicamente come numero e fuori contesto.
Infine, i testi, nella forma attuale, devono essere normalizzati per avere una forma non ambigua e facilitare la fase di training.

4.2 Data cleaning

4.2.1 Valori numerici

Le prime features del processo di pulizia sono `upload_hour`, `likes`, `dislikes` e `comments`. In esse, abbiamo due problematiche principali:

- La presenza di valori mancanti;
- La presenza di valori assurdi/anomali.

La prima problematica si risolve con una semplice **Imputazione statistica**, usando come funzione la Mediana.

```
import pandas as pd
import re
# ...NLP libraries
def clean_dataset(df):
    def value_imputation(df, column):
        median_value = df[column].median()
        df[column].fillna(median_value, inplace=True)
        return df
    for col in ['upload_hour', 'likes', 'dislikes', 'comments']:
        df = value_imputation(df, col)
```

La seconda problematica si risolve utilizzando una funzione statistica di **Range Interquartile** (IQR), i valori in analisi che non rispettano il suddetto vengono rimpiazzati con il valore mediano della colonna in cui esso si trova.

$$\text{IQR} = Q_3 - Q_1$$

dove

$$Q_1 = \text{termine in posizione } \frac{n+1}{4}$$

$$Q_3 = \text{termine in posizione } \frac{3(n+1)}{4}$$

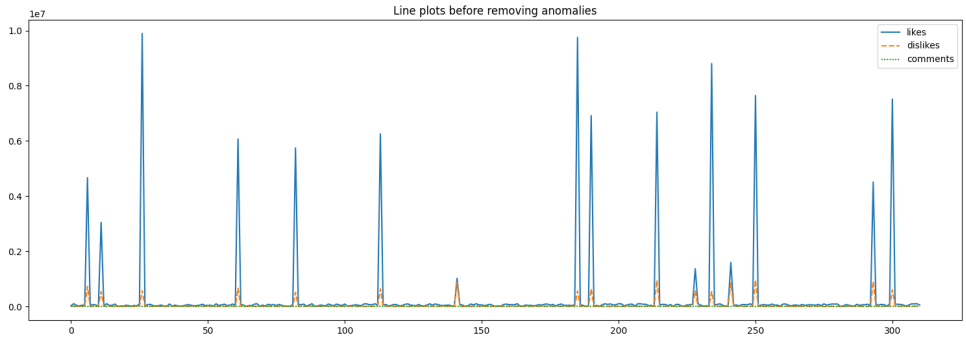


Figure 4.1: Prima della rimozione degli Outliers (anomalie)

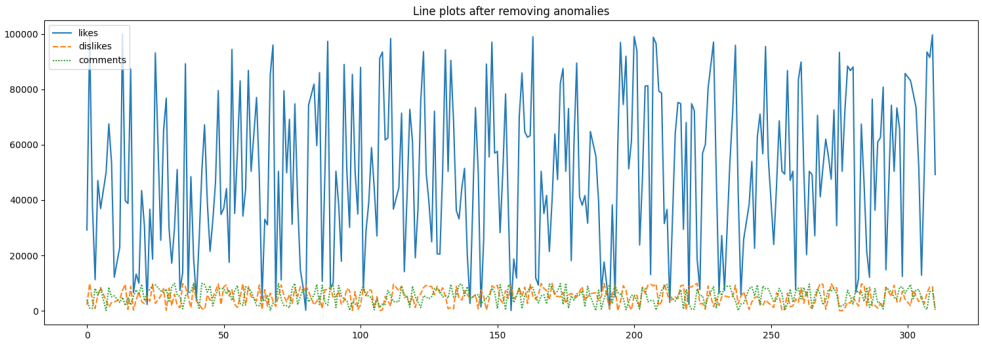


Figure 4.2: Dopo la rimozione degli Outliers (anomalie)

```python
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from spellchecker import SpellChecker
from num2words import num2words

def clean_dataset(df):
    # ...
    def remove_anomalies(df, column):
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        median_val = df[column].median()
        df.loc[df[column] < lower_bound, column] = median_val
        df.loc[df[column] > upper_bound, column] = median_val
        return df
    for col in ['likes', 'dislikes', 'comments']:
        df = remove_anomalies(df, col)
```

4.2.2 Valori testuali

Le successive features soggette al processo di pulizia sono le feature testuali, ovvero `title` e `description`.

In esse abbiamo 4 problematiche principali:

- La presenza di valori di errore ("###ERROR###");
- La presenza di caratteri speciali;
- Testo non normalizzato;
- La presenza di valori corrotti rimanenti dopo i precedenti processi (solo `title`).

La prima problematica (valori di errore) è risolvibile facilmente, eseguendo un parsing con le istanze che rispettano il criterio di eguaglianza con la stringa:

```
import # ...

def clean_dataset(df):
    # ...
    def clean_text(text):
        if pd.isnull(text) or '###ERROR###' in str(text):
            return "Title Not Available"

    df['title'] = df['title'].apply(clean_text)
```

La seconda problematica (caratteri speciali) è ancor piu' facile da risolvere, rimuovendo semplicemente i caratteri speciali utilizzando delle **Regular Expression**.

```
import # ...

def clean_dataset(df):
    # ...
    def clean_text(text):
        # ...
        text = re.sub(r'[^\w\s]', '', str(text))
        text = re.sub(r'\s+', ' ', text).strip()

        return text

    df['title'] = df['title'].apply(clean_text)
```

La prima REGEX serve a eliminare i caratteri speciali, la seconda elimina gli spazi quando ce ne sono più di uno.

Successivamente viene eseguita l'eliminazione delle istanze corrotte riconosciute fino a questo punto.

```
import # ...

def clean_dataset(df):
    # ...
    def clean_text(text):
        # ...

    df = df[df['title'] != "Title Not Available"]
```

La terza problematica (testo non normalizzato) si può risolvere seguendo delle regole standard per la normalizzazione del testo. In questo caso sono state praticate, per entrambi titolo e descrizione:

- Contraction Expansion;
- Tokenizzazione;
- Conversione dei numeri (in cifre) in parole;
- Lemmatizzazione;
- Trasformazione in Minuscolo;
- Stopword Removal.

```
import # ...

def clean_dataset(df):
    # ...
    def clean_text(text):
        # ...
    def normalize_text(text):
        if not isinstance(text, str):
            print(f"Titolo non valido: {text}")
            return None

        try:
            contractions = {
                "I'm": "I am", "you're": "you are", "he's": "he is",
                "she's": "she is", "it's": "it is", "we're": "we are",
                "they're": "they are", "can't": "cannot",
                "won't": "will not", "don't": "do not",
                "didn't": "did not", "isn't": "is not",
            }
            for contraction, full_form in contractions.items():
                text = re.sub(r'\b' + contraction + r'\b', full_form,
    text)

            tokens = word_tokenize(text)
```

```
24
25              tokens = [
26                  num2words(word) if word.isdigit() else word
27                  for word in tokens
28              ]
29
30              tokens = [lemmatizer.lemmatize(word) for word in tokens if
    word]
31
32              tokens = [word.lower() for word in tokens]
33
34              tokens = [word for word in tokens if word not in stop_words
    ]
35
36              normalized_text = ' '.join(tokens)
37
38              return normalized_text if normalized_text.strip() else None
39          except Exception as e:
40              print(f"Errore durante la normalizzazione del titolo: {text
    }. Errore: {e}")
41              return None
42
43      df['title'] = df['title'].apply(normalize_text)
```

Anche qui si controlla se il titolo risultante è vuoto, e nel caso si elimina la riga corrispondente.

La quarta problematica (valori corrotti) invece può essere risolta solo manualmente, non esiste un sistema automatico funzionante al 100% in quanto i valori corrotti sono di natura totalmente casuale.
Andiamo dunque a eliminare istanze del nuovo dataset risultato dalle precedenti operazioni che contengono valori corrotti. Dall'osservazione fatta, risultano essere i seguenti (elencati per titolo):

- "pqjjlbwwsupivcnmfi9i2m8jtdw"
- "son0mmgo6vr2rbo6ixhnjnvvem2r8qc"

Per quanto riguarda la descrizione, eseguiremo tutti i passaggi precedenti anche per essa, tenendo in conto che i link presenti non devono essere modificati, perderebbero senò di significato.

```
import # ...

def clean_dataset(df):
    # ...
    def clean_text(text):
        # ...
    def normalize_text(text):
        # ...

    def description_case(desc):
        if pd.isnull(desc):
            return None

        link_pattern = r'(http[s]?://\S+)'
        match = re.search(link_pattern, desc)
        link = match.group(1) if match else ''

        text_part = re.sub(link_pattern, '', desc).strip()

        cleaned_text = clean_text(text_part)

        normalized_text = normalize_text(cleaned_text) if cleaned_text
    else None

        final_description = (normalized_text or '').strip()
        if link:
            final_description = (final_description + ' ' + link).strip
    ()

        return final_description

    df['description'] = df['description'].apply(description_case)
```

4.2.3 Feature Scaling

I valori numerici attuali hanno significato solo nella propria feature, ma se comparati con le altre avranno poco significato contestuale, e più di semplici numeri. Il modello assumerà che una feature sarà più importante di un'altra. Dunque, normalizziamo questi valori usando la **Min-Max Normalization**.

$$x' = a + \frac{(x - min(x)) \cdot (b - a)}{max(x) - min(x)}$$

Dove x' è il valore da immettere al posto dell'attuale, risultato della formula, $a = 0$, ovvero il minimo della nostra scala di normalizzazione, $b = 1$, ovvero il massimo della nostra scala di normalizzazione.
Per non rendere i valori difficili da analizzare, viene fatto anche un arrotondamento di due cifre decimali.

L'unica eccezione di questo processo è la feature `upload_hour`, che ha un significato assoluto che verrebbe eliminato se i suoi valori venissero normalizzati. Dunque i suoi valori rimarranno gli stessi.

```
1 def scale_column(df, column):
2     df[column] = (df[column] - df[column].min()) / (df[column].max() -
    df[column].min())
3 
4     df[column] = df[column].round(2)
5 
6     return df
```

4.2.4 Feature Selection

Dobbiamo ora analizzare le feature e capire se sono tutte utili alla risoluzione del problema, tenendo conto del dominio di quest'ultimo e delle conoscenze acquisite fin'ora su di esso.

Sappiamo che i video truffaldini possono essere **sicuramente** riconosciuti tramite alcune parole chiave, utilizzate molto spesso per la loro facile attrazione ("Se qualcosa è troppo bello per essere vero, allora probabilmente lo è").

Dunque, le features `title` e `description` saranno essenziali alla risoluzione del problema.

Volendo migliorare codesta soluzione, possiamo dividere la descrizione e il link, contenuto in essa, in due features diverse:

- `description`, che manterrà il contenuto (normalizzato) testuale della descrizione;
- `link_desc`, che prenderà come valori "no link" in caso non sia presente un link, e la stringa del link nel caso essa sia invece presente.

```
import pandas as pd
import re

def feature_selection(df):
    df_new = df.copy()

    def extract_link(text):
        if pd.isnull(text):
            return "No link"
        match = re.search(r'http[s]?://\S+', text)
        return match.group() if match else "no link"

    df_new['link_desc'] = df_new['description'].apply(extract_link)

    def remove_link(text):
        if pd.isnull(text):
            return text
        return re.sub(r'http[s]?://\S+', '', text).strip()

    df_new['description'] = df_new['description'].apply(remove_link)
```

Ora, avendo una feature completamente diversa per il valore delle stringhe "link", possiamo pulire e normalizzare il testo, come già fatto per il processo di data cleaning del titolo, avendo ora, il valore, un contesto proprio.

```
import # ...

def feature_selection(df):
    def extract_link(text):
    # ...
    def remove_link(text):
    # ...

    def clean_link_desc(link):
        if not isinstance(link, str) or not link:
            return link
        text = re.sub(r'[^\w\s]', ' ', link)
        text = re.sub(r'\s+', ' ', text).strip()
        return text

    df_new['link_desc'] = df_new['link_desc'].apply(clean_link_desc)
```

Per quanto riguarda le features con valori numerici, tramite un analisi del dominio si può facilmente capire che sono informazioni non utili alla risoluzione del problema, in quanto i valori stessi dipendono dalle interazioni che il pubblico ha con il suddetto contenuto, non dalla veridicità del contenuto.

Eliminiamo dunque le features numeriche, avendo natura casuale e indipendente.

```
import # ...
def feature_selection(df):
    def extract_link(text):
    # ...
    def remove_link(text):
    # ...
    def clean_link_desc(link):
    # ...
    df_new.drop(['upload_hour','likes','dislikes','comments'], axis=1,
   inplace=True)
    return df_new
```

4.2.5 Data Balancing

Il dataset, dopo una nemmeno troppo accurata analisi, risulta essere ben equilibrato, non ci sono carenze per nessuna delle due classi presenti nel problema.

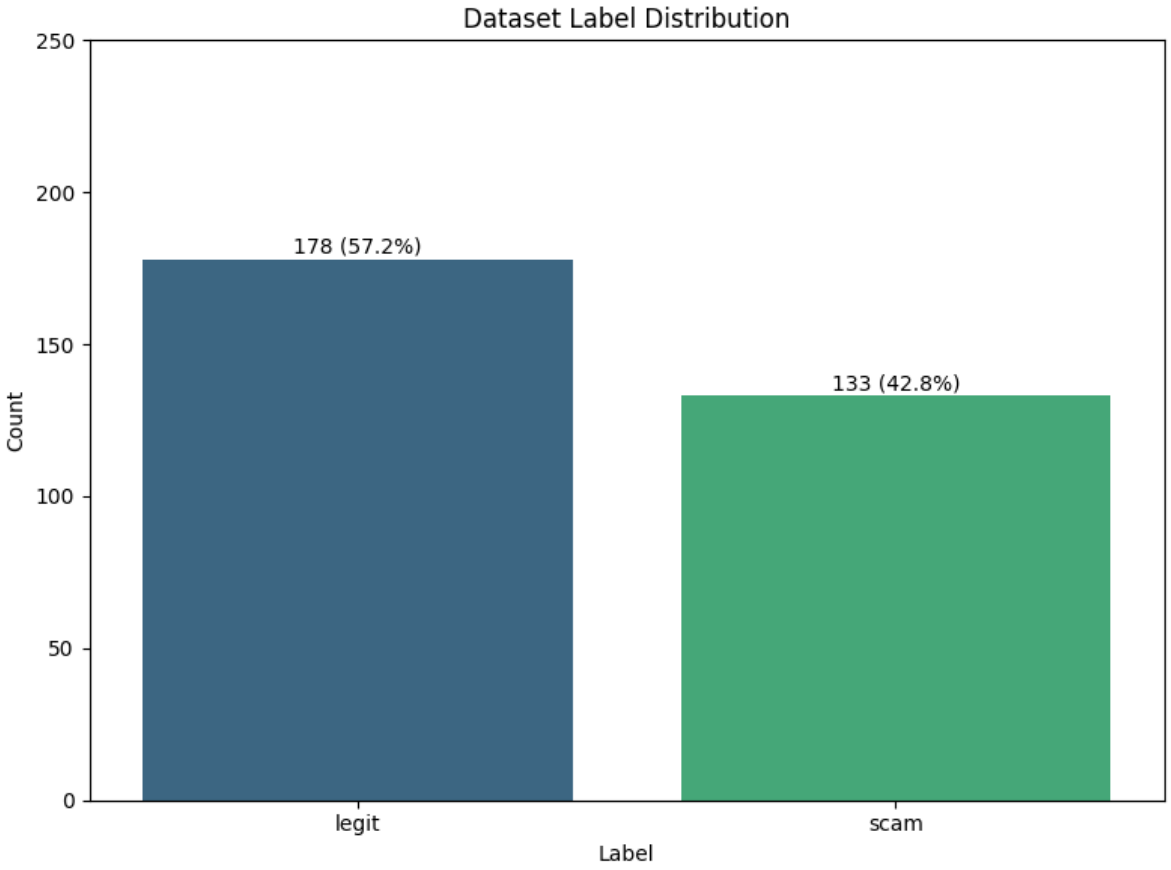


Figure 4.3: Equilibrio delle istanze del dataset

CHAPTER 5

Data Modeling

5.1 Introduzione al Modellamento

In questa fase costruiremo il nostro modello, seguendo due fasi ben precise:

- Scelta di tecniche e algoritmi;
- Addestramento del Machine Learning Model.

5.2 Tecnica

In questa fase, analizziamo il problema in esame per capire con quale tecnica creare il nostro Modello.
Essendo un problema in cui abbiamo bisogno di etichettare le nuove istanze che verranno date al Modello, e i dati stessi di traning sono anche essi etichettati, possiamo dedurre che questo problema è un problema di **Classificazione**, più nello specifico un problema di classificazione binaria, in quanto le etichette possono assumere solo due valori: "scam" o "legit".

5.3 Algoritmi

Per un problema di classificazione abbiamo a disposizione un gran numero su cui fare affidamento.
Tra di essi ne scegliamo due, in quanto vogliamo comparare quali dei due potrebbe darci migliori performance in questo ambiente:

- `Naïve Bayes`;
- `Random Forest`.

Ora, andiamo ad analizzare perchè dovremmo preferire uno dei due, all'altro.

5.3.1 Naïve Bayes

L'algoritmo Naïve Bayes è uno dei classificatori più semplici e veloci, sia in fase di addestramento che di utilizzo, grazie alla sua efficienza computazionale e al presupposto di indipendenza tra le feature.

Questo lo rende particolarmente adatto per problemi di classificazione con dataset piccoli o medi, dove la velocità e l'usabilità sono prioritarie.
Tuttavia, un limite significativo di Naïve Bayes è la sua assunzione di indipendenza tra le feature, che raramente è valida per dati reali. Nel contesto del problema affrontato da OverseerAI, questa limitazione è particolarmente rilevante: le feature `title`, `description` e `link_desc` mostrano correlazioni evidenti e devono essere analizzate in relazione l'una con l'altra per produrre risultati significativi.
Di conseguenza, pur essendo un algoritmo rapido ed efficiente, Naïve Bayes potrebbe non essere la scelta ottimale per problemi in cui le dipendenze tra le feature giocano un ruolo cruciale nella qualità della classificazione.

5.3.2 Random Forest

L'algoritmo Random Forest che è stato scelto per affrontare il problema di classificazione come algoritmo principale grazie alla sua capacità di gestire dataset con feature correlate.
Esso costruisce un insieme di alberi decisionali e combina le loro previsioni, migliorando così la robustezza e riducendo il rischio di overfitting rispetto a un singolo albero decisionale.
Grazie all'uso di più alberi decisionali, esso riesce anche a trovare pattern nascosti, qualità che nell'attuale problema in analisi ha grande rilevanza e utilità.
Queste caratteristiche hanno determinato come scelta definitiva del suddetto algoritmo per la risoluzione del problema.
Anche se esso richiede più risorse computazionali rispetto a algoritmi più semplici, come Naïve Bayes, il bilanciamento tra accuratezza e capacità di generalizzazione (utile ai pattern nascosti) lo rende una scelta ideale per il problema in esame.

5.4 Addestramento

Andiamo ora ad addestrare il nostro modello, usando l'algoritmo Random Forest.
Per praticità di utilizzo, e per fornire una demo riproducibile, il modello addestrato verrà salvato in formato .pkl, per mantenere tutto il settaggio del modello eseguito in fase di training.

Come primo step, scriviamo il codice che processerà il dataset e allenerà il nostro modello:

```
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
   recall_score, f1_score
import joblib

def create_pipeline():
    preprocessor = ColumnTransformer(
        transformers=[
        ('title_tfidf', TfidfVectorizer(max_features=5000), 'title'),
        ('description_tfidf', TfidfVectorizer(max_features=5000), '
   description'),
        ('link_desc_tfidf', TfidfVectorizer(max_features=5000), '
   link_desc')
        ]
    )

    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', RandomForestClassifier(n_estimators=100,
   random_state=42, class_weight="balanced"))
    ])

    return pipeline
```

Prima di iniziare ad addestrare il modello, abbiamo bisogno di processare i dati per renderli "capibili" al nostro classificatore, se ne occuperà il `preprocessor`.

In `ColumnTransformer()` trasformiamo le nostre colonne (features) in vettori numerici, utilizzando la funzione **TF-IDF** (Term Frequency-Inverse Document Frequency), che quantifica l'importanza di una parola all'interno del dataset. Il dizionario di importanza nella nostra soluzione avrà un valore arbitrario massimo di 5000 elementi.

In `pipeline()` utilizziamo la funzione Pipeline per creare una "catena di montaggio", che eseguirà in ordine questi step:

1. Esecuzione del `preprocessor`;
2. Esecuzione del classificatore `RandomForestClassifier()` con 100 alberi differenti, seed randomico (con cui si possono replicare i risultati) = 42 e un compensamento automatico per lo squilibrio delle classi nel problema di esame, dando un peso maggiore alle classi meno rappresentate.

La successiva fase utilizzerà ciò che abbiamo creato per allenare il modello, e validerà ogni istanza di allenamento eseguita.

```
import # ...

def train_and_evaluate(df):
    X = df[['title', 'description', 'link_desc']]
    y = df['label']

    kf = KFold(n_splits=10, shuffle=True, random_state=42)
    fold_num = 1

    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores = []
    confusion_matrices = []

    y_true_all = []
    y_pred_all = []

    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        pipeline = create_pipeline()
        pipeline.fit(X_train, y_train)
        y_pred = pipeline.predict(X_test)
```

```
26
27          cm = confusion_matrix(y_test, y_pred)
28          confusion_matrices.append(cm)
29
30          y_true_all.extend(y_test)
31          y_pred_all.extend(y_pred)
32
33          acc = accuracy_score(y_test, y_pred)
34          prec = precision_score(y_test, y_pred, average='macro')
35          rec = recall_score(y_test, y_pred, average='macro')
36          f1 = f1_score(y_test, y_pred, average='macro')
37
38          print(f"Fold {fold_num} - Accuracy: {acc:.3f}, Precision: {prec
    :.3f}, Recall: {rec:.3f}, F1-Score: {f1:.3f}")
39
40          accuracy_scores.append(acc)
41          precision_scores.append(prec)
42          recall_scores.append(rec)
43          f1_scores.append(f1)
44
45          fold_num += 1
46
47      # ...
48
49      joblib.dump(pipeline, 'random_forest_model.pkl')
50      print("Modello salvato come 'random_forest_model.pkl'")
```

I dati che vengono usati per l'allenamento vengono presi dal dataset e divisi in due categorie: una parte va al training, l'altra va alla validazione della fase di training.

Per questo processo usiamo il metodo di convalida **K-Fold N-times cross validation**. Nel nostro caso `N-times = 10`, e la funzione mescola le istanze del dataset prima di dividerle in gruppi per evitare dipendenze sequenziali dei dati (e viene fornito un seed per replicare l'istanza di training). Poi, per ogni fold, usa il 90% del fold per il training, e il restante 10% per la validazione.

Ad ogni iterazione del ciclo `for` viene eseguita la singola iterazione del `K-fold`.

Sapendo che **X** sono le istanze con valori title, description e link_desc, e **y** sono le istanze con solo label come valore, all'interno del ciclo vengono assegnati valori di training e

test per X e valori di training e test per y, utilizzando lo split attuale (creato usando `kf.split(X)`).

Si procede, dopo questa suddivisione, a creare la pipeline di preprocessamento vista in precedenza, e con `pipeline.fit()` allena il modello.

`y_pred = pipeline.predict(X_text)` cerca invece di predirre ogni valore di X_text.

Successivamente, la funzione calcola le metriche di valutazione per:

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

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

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

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Dove si usano i valori dei True Positives (TP), True Negatives (TN), False Positives (FP) e False Negatives (FN).

L'intenzione è massimizzare codesti valori.

Infine, dopo il completamento dell'esecuzione del ciclo `for`, utilizziamo la funzione `joblib.dump()` per salvare il traning e la validazione fatti come modello allenato.

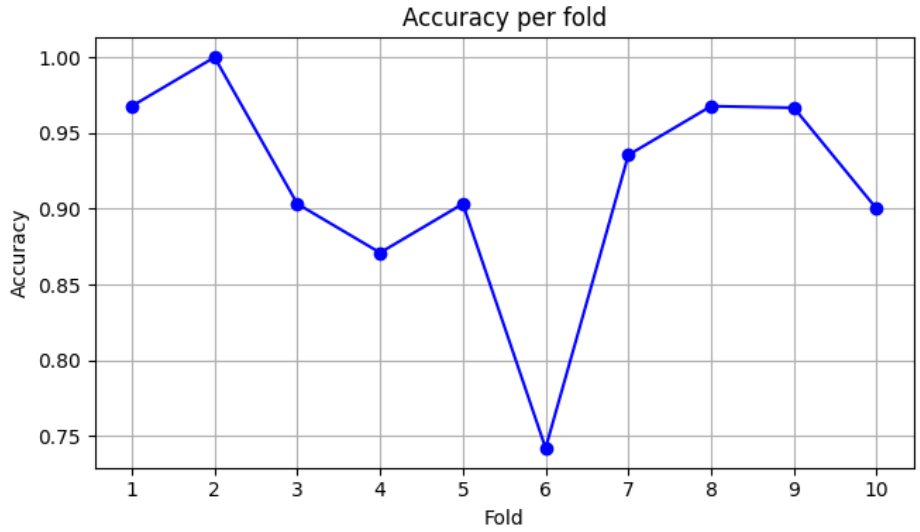


Figure 5.1: Grafico Accuracy

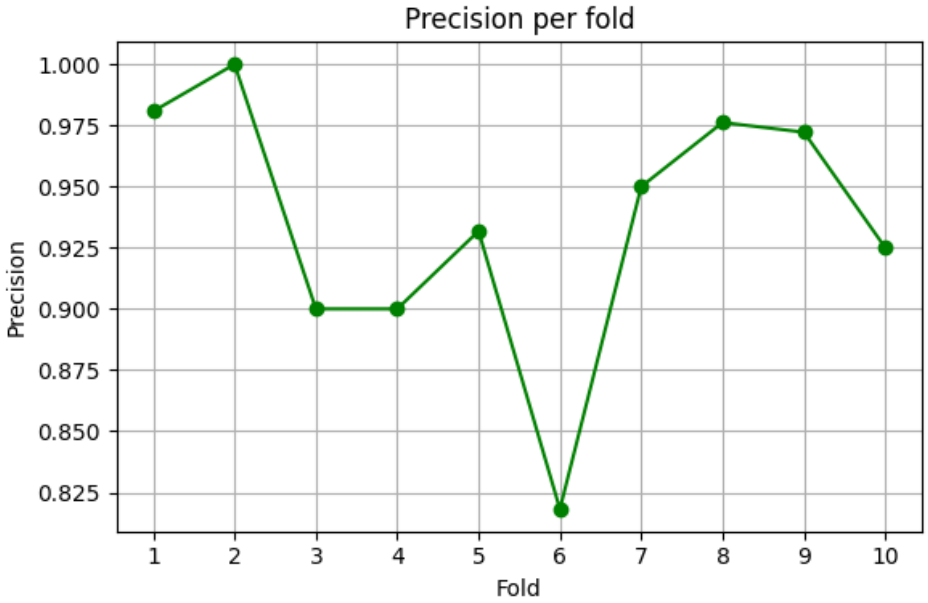


Figure 5.2: Grafico Precision

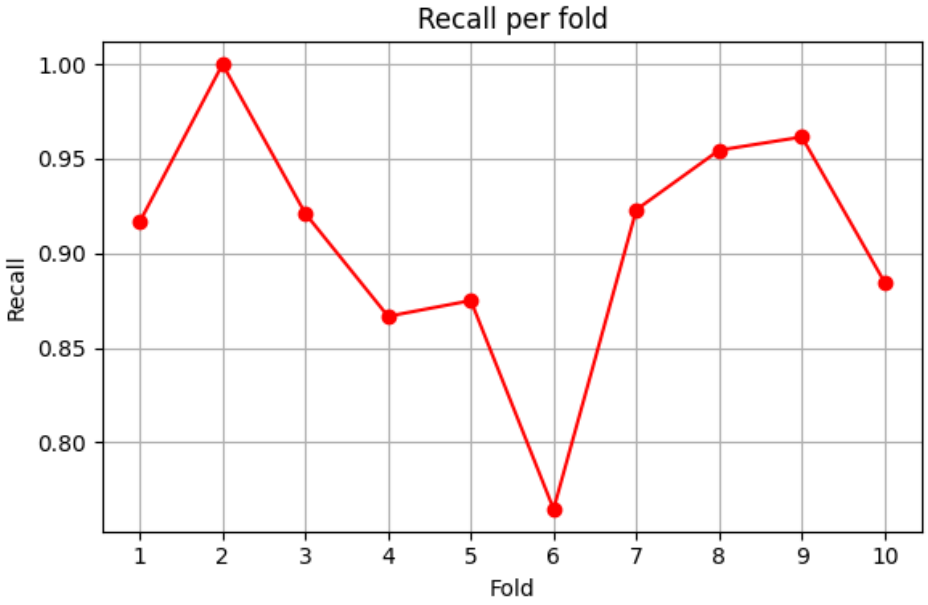


Figure 5.3: Grafico Recall

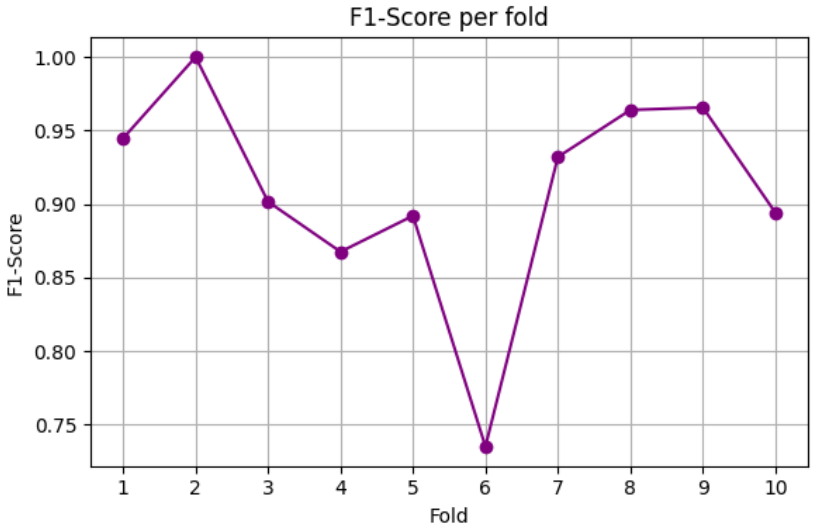


Figure 5.4: Grafico F1-Score

CHAPTER 6

Evaluation

6.1 Valutazione

Confusion Matrix

Durante l'esecuzione della fase di Training il modello è stato adattato ulteriormente con l'aggiunta del calcolo della **Matrice di Confusione**, così da poter meglio valutare i risultati.
Le Matrici risultanti sono:

- Una per iterazione (per fold);

Fold	TN	FP	FN	TP
Fold 1	25	0	1	5
Fold 2	18	0	0	13
Fold 3	12	0	3	16
Fold 4	16	0	4	11
Fold 5	19	0	3	9
Fold 6	14	0	8	9
Fold 7	18	0	2	11
Fold 8	20	0	1	10
Fold 9	17	0	1	12
Fold 10	17	0	3	10

- Una finale di valutazione globale.

TN	FP	FN	TP
176	0	26	106

Dalle Matrici ci risulta essere evidente che il modello non è perfetto, sicuramente è migliorabile, ma è anche decisamente nella norma e utilizzabile.
Nella fase di **Deployment**, testeremo tramite uso reale del modello la sua usabilità.

Metriche Accuracy, Precision, Recall e F1-Score

Valori per-fold:

Fold	Accuracy	Precision	Recall	F1
1	0.968	0.981	0.917	0.945
2	1.000	1.000	1.000	1.000
3	0.903	0.900	0.921	0.902
4	0.871	0.900	0.867	0.868
5	0.903	0.932	0.875	0.892
6	0.742	0.818	0.765	0.735
7	0.935	0.950	0.923	0.932
8	0.968	0.976	0.955	0.964
9	0.967	0.972	0.962	0.966
10	0.900	0.925	0.885	0.894

Valori globali medi:

Accuracy	Precision	Recall	F1
0.916	0.935	0.907	0.910

Come possiamo notare i valori delle metriche di valutazione risultano essere eccezionali. Al secondo Fold addirittura uno score di 1 su tutte le metriche!
Ovviamente ci affideremo solo in parte ad esse, ma il loro valore > 0.8 è decisamente un ottimo risultato!
Un dettaglio che possiamo notare sia dalla Confusion Matrix che dalle metriche di valutazione è che nel Fold N.6 il training subisce una perdita enorme, con ben 8 False Negatives.
Un aspetto su cui si potrebbe ritornare per migliorare il modello potrebbe essere anche questo.

6.2 Valutazione post-Deployment

Andremo qui ad analizzare la problematica riscontrata nell'istanza:

6.2.1 Istanza legit con link

Da un'analisi più approfondita, risulta essere un problema già osservato nella Matrice di Confusione vista in precedenza: vi sono un numero cospicuo di Falsi Negativi (parlando proporzionalmente al dataset).
Questo è un esempio di Falso Negativo.
Provando diverse combinazioni, si scopre che l'istanza viene erroneamente classificata per via di un Bias creatosi durante il training del modello per la feature "link_desc".
Infatti, se andiamo a provare la stessa istanza usata in questo test, ma togliendo il link, verrà classificata nel modo giusto.

```
Inserisci il titolo:
Top 10 coding languages to learn
Inserisci la descrizione:
Today we are going to cover the top coding languages to learn for yourself
Risultato della classificazione: LEGIT
```

Questo problema potrebbe essere risolto dando un peso minore in fase di allenamento alla feature "link_desc", rispetto alle altre.
Questa soluzione è stata realizzata, ma non è stata completata per mancanza di tempo.

CHAPTER 7

Deployment

7.1 Realizzazione Interfaccia

In questa fase, verrà semplicemente realizzata un'interfaccia tramite terminale per usare e provare il Modello.

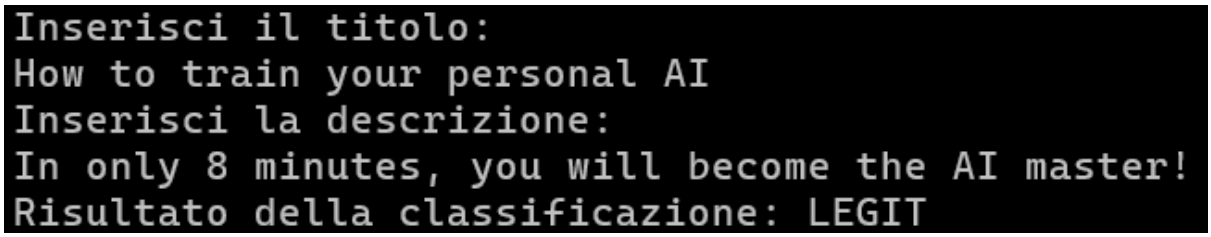
```
import joblib
import pandas as pd
import re
import nltk
from nltk # ...
def normalize_text(text):
    # Funzione di normalizzazione come visto nel Data Cleaning
def extract_link(text):
    # Funzione di estrazione come visto nel Feature Selection
def load_model(filepath):
    return joblib.load(filepath)

def predict_title(model, title, description):
    normalized_title = normalize_text(title)
    desc_text_part, desc_link = extract_link(description)
    normalized_description = normalize_text(desc_text_part)
    normalized_link_desc = normalize_text(desc_link)
    input_data = pd.DataFrame([{
        'title': normalized_title,
        'description': normalized_description,
        'link_desc': normalized_link_desc
    }])
    prediction = model.predict(input_data)
    return prediction[0]

if __name__ == "__main__":
    model_path = "../3_data_modeling/random_forest_model.pkl"
    model = load_model(model_path)
    print("Inserisci il titolo:")
    title = input()
    print("Inserisci la descrizione:")
    description = input()
    result = predict_title(model, title, description)
    print(f"Risultato della classificazione: {result.upper()}")
```

7.2 Test di Usabilità

Per il nostro test proveremo 4 istanze separate (basate sulla conoscenza del dominio):

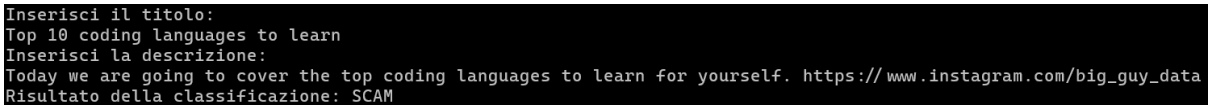
- Istanza legit senza link;
- Istanza legit con link;
- Istanza scam senza link;
- Istanza scam con link.

7.2.1 Istanza legit senza link



L'istanza viene riconosciuta nel modo giusto, quindi passa questo test.

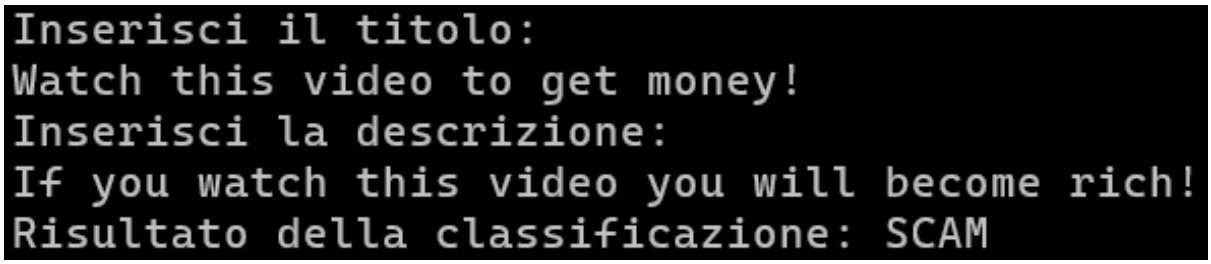
7.2.2 Istanza legit con link



L'istanza viene riconosciuta nel modo sbagliato, quindi non passa questo test.

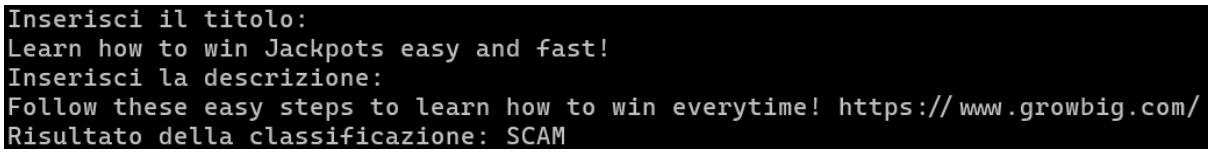
Nella fase di Evaluation verrà inserita un'analisi su questa casistica.

7.2.3 Istanza scam senza link



L'istanza viene riconosciuta nel modo giusto, quindi passa questo test.

7.2.4 Istanza scam con link



L'istanza viene riconosciuta nel modo giusto, quindi passa questo test.

CHAPTER 8

Conclusioni

8.1 Conclusioni finali

La realizzazione di questo progetto è stata divertente e molto formativa, nonostante il corso fosse di "Fondamenti" sento di aver superato lo scopo del corso per il bene della mia formazione, e spero che questo progetto possa invogliare altri (possibili) lettori a fare lo stesso.