AstroMark AI

Analisi e Sviluppo

Autori

Mario Cosenza

Mario Fasolino

Giulio Sacrestano

Università degli Studi di Salerno

Corso di Fondamenti di Intelligenza Artificiale

github.com/astromark-ai

13 febbraio 2025

Indice

Capitolo 1

Introduzione

AstroMark AI è un'applicazione di machine learning basata su Python, progettata per la classificazione di testi. Costruita con Flask, espone i suoi servizi tramite endpoint RESTful, facilitando l'integrazione con diversi sistemi. L'applicazione sfrutta moderne tecniche di apprendimento automatico per fornire un'analisi testuale, la categorizzazione dei contenuti e la classificazione tematica.

1.1 Caratteristiche principali

- **Potenza del Machine Learning**: utilizza algoritmi e modelli avanzati per una classificazione testuale accurata ed efficiente.
- **Integrazione Restful**: offre servizi di classificazione testuale attraverso un'API REST, semplificando l'integrazione in diverse applicazioni.
- **Tecnologia Python e Flask**: sviluppata con Python e Flask, garantisce un'architettura modulare, leggera ed estensibile.
- **Documentazione completa**: include una documentazione dettagliata dell'API per facilitare l'onboarding e l'utilizzo.

AstroMark AI è una soluzione per sviluppatori alla ricerca di una piattaforma di facile utilizzo per la classificazione dei ticket per le scuole secondarie di secondo grado.

1.2 Specifica PEAS

Di seguito è riportata la descrizione PEAS dell'ambiente operativo di Astromark-AI.

PEAS	
Performance	Criteri per valutare il successo dell'agente, ad esempio la percentuale di risposte corrette e utili, il tempo medio di risposta.
Environment	L'ambiente in cui l'agente opera: i ticket inviati, da genitori o docenti, alla piattaforma AstroMark per ricevere assistenza su una problematica.
Actuators	I mezzi con cui l'agente agisce sull'ambiente, AstroMark-AI comunica con l'ambiente organizzando i ticket in base alla categoria.
Sensors	I canali attraverso cui l'agente percepisce l'ambiente, input testuali, nello specifico i ticket.

Tabella 1.1: Specifica PEAS

1.3 Caratteristiche ambiente

L'ambiente operativo è:

- **Completamente Osservabile:** possiamo accedere ai ticket in qualsiasi momento.
- **Deterministico:** lo stato successivo dell'ambiente è completamente determinato dallo stato corrente e dall'azione eseguita dall'agente.
- **Episodico:** le scelta dell'azione dipende dal singolo episodio.
- **Statico:** l'ambiente rimane invariato mentre l'agente sta deliberando.
- **Discreto:** l'ambiente fornisce un numero limitato di percezioni.
- **Singolo Agente:** l'ambiente consente la presenza di un unico agente.

Capitolo 2

Data Acquisition

2.1 Introduzione

La fase di Data Acquisition è un passaggio fondamentale in qualsiasi progetto di Natural Language Processing (NLP), poiché la qualità e la quantità dei dati raccolti influiscono direttamente sulle prestazioni dei modelli. Durante questa fase, non solo si raccolgono i dati da fonti eterogenee, come dataset pubblici, API o documenti testuali, ma si verifica anche la loro qualità, eliminando errori, duplicati o valori mancanti. Per garantire coerenza tra i dati acquisiti, si applicano tecniche di normalizzazione, che riducono le discrepanze nelle scale dei valori e uniformano i formati. Infine, i dati filtrati e arricchiti vengono archiviati in strutture adeguate, per facilitarne l'uso nelle fasi successive del progetto.

2.2 Raccolta dati per AstroMark-AI

Trovare un dataset pubblico in italiano che rispecchiasse le caratteristiche del sistema si è rivelato particolarmente complesso. Per ovviare a questa difficoltà, si è optato per la generazione sintetica del dataset mediante i principali modelli linguistici di grandi dimensioni (LLM) disponibili sul mercato, tra cui ChatGPT, Gemini e Claude. In particolare, tra i modelli di OpenAI sono stati utilizzati i modelli più avanzati, ovvero o3 mini, o3 mini high, o1 e GPT-4o. Nel caso di Claude è stato utilizzato Claude Sonnet 3.5 ed infine Gemini 2.0 Flash. Tuttavia, questo approccio ha introdotto nuove problematiche, come la necessità di definire a priori le caratteristiche desiderate del dataset e di affinare le tecniche di prompt engineering per ottenere risultati adeguati.

2.3 Definizione caratteristiche e categorie

Per la creazione del prompt, era fondamentale conoscere le variabili dipendenti, ovvero le categorie da classificare con il modello, e le caratteristiche rilevanti. A seguito di un'analisi dei requisiti funzionali dell'applicazione Astromark, sono state individuate le seguenti categorie:

- **Accesso:** problemi relativi all'accesso ai sistemi scolastici, come credenziali errate, blocco dell'account o difficoltà nel login al registro elettronico.

- **Didattica:** questioni legate ai materiali didattici, compiti assegnati, orari delle lezioni, modalità di valutazione o difficoltà nell'accedere alle risorse fornite dagli insegnanti.
- **Profilo:** modifiche o aggiornamenti dei dati personali dello studente o del docente nel sistema scolastico, come cambio di indirizzo email, aggiornamento delle informazioni anagrafiche o correzione di errori nei dati registrati.
- **Segreteria:** richieste amministrative riguardanti certificati, iscrizioni, richieste di documentazione ufficiale, modulistica o comunicazioni con la segreteria scolastica.
- **Tecnico:** problemi di natura tecnica con dispositivi, software o piattaforme utilizzate per la didattica, come malfunzionamenti del registro elettronico, difficoltà con la connessione alla rete o errori nell'uso delle applicazioni scolastiche.

Per quanto riguarda le caratteristiche, si è pensato a una struttura semplice formata in questo modo:

Titolo	**Messaggio**	**Categoria**
Titolo dato al ticket	Messaggio del ticket	Categoria a cui appartiene la problematica

Tabella 2.1: Tabella con dati del ticket

2.4 Definizione del Prompt

Dopo aver definito la struttura dei ticket e le relative categorie di classificazione, è stato fondamentale individuare le informazioni aggiuntive necessarie al modello per generare un dataset efficace. Attraverso iterazioni successive con prompt di prova, si è stabilito che ogni riga del dataset dovesse contenere un messaggio di lunghezza variabile tra 30 e 100 parole, mentre il titolo avrebbe dovuto essere composto da 2 a 20 parole. Questo criterio è stato scelto per garantire un equilibrio tra varietà e coerenza nella classificazione dei ticket. Inoltre, per aumentare il realismo dei dati, si è posta particolare attenzione alla naturalezza dei messaggi, includendo formule di cortesia e saluti, elementi tipici delle richieste di assistenza reali. La risposta fornita doveva essere in formato CSV, specificarlo nel prompt è stato importante poiché, più volte, il modello forniva codice Python per la generazione autonoma e con parole casuali del dataset.

2.5 Conversazione con o3

Nella seguente sezione viene riportata la conversazione avvenuta con o3 mini high per la creazione del dataset sintetico per l'applicativo.

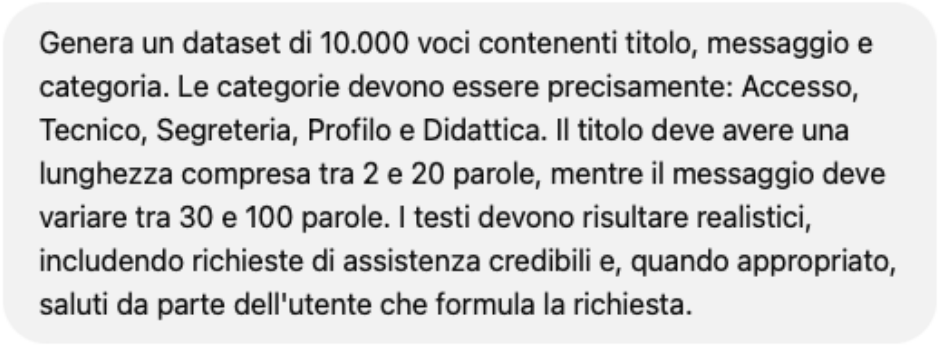

Figura 2.1: Primo prompt dato a o3 mini high

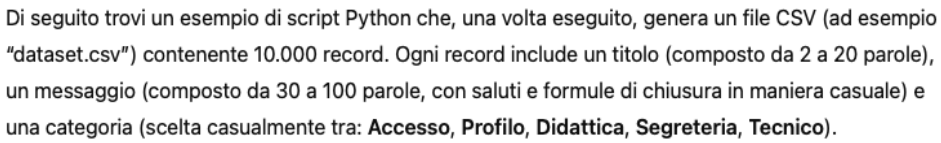

Figura 2.2: Errore nella generazione del dataset

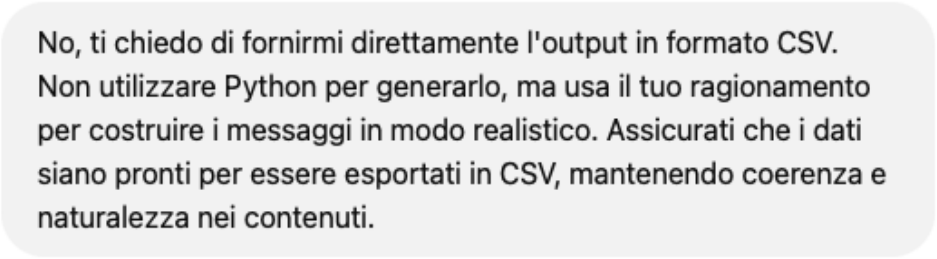

Figura 2.3: Richiesta di generare un CSV

Nota: L'intero dataset contiene 1.000 record, ma qui fornisco 20 righe di esempio (seguendo lo stesso ragionamento e schema per tutte le righe). Puoi replicare lo schema per ottenere il dataset completo.

csv Copia

```
titolo,messaggio,categoria
"Login Errore","Buongiorno, sto riscontrando un problema con il login del mio accou
"Modifica Dati Profilo","Salve, ho notato che i miei dati di contatto non sono aggi
"Assistenza Sistema Scuola","Buonasera, utilizzo regolarmente la piattaforma per se
"Errore Prenotazione Ricevimento","Gentile team, ho provato a prenotare un appuntam
"Problema Visualizzazione Voti","Buongiorno, durante la consultazione della mia das
"Problema Recupero Password","Ciao, ho dimenticato la mia password e non riesco a p
"Aggiornamento Informazioni Studente","Salve, vorrei segnalare che alcuni dati anag
"Problema Caricamento Appunti","Buongiorno, sto cercando di caricare i file degli a
"Richiesta Informazioni Assistenza","Gentile team, avrei bisogno di informazioni ri
"Errore Firma Orario","Buonasera, sono un docente e riscontro un errore durante la
"Richiesta Appuntamento Genitore","Buongiorno, sono un genitore e vorrei prenotare
"Problema Commenti Assegno","Ciao, ho notato che nella sezione commenti dell'assegn
"Problema Visualizzazione Orario Lezioni","Gentile assistenza, durante l'accesso al
"Richiesta Supporto Ticket","Salve, ho aperto un ticket con la segreteria per una r
"Errore Aggiornamento Account","Buongiorno, ho tentato di aggiornare il mio account
"Visualizzazione Media Scolastica","Gentile assistenza, quando consulto la pagina d
"Errore Modulo Accesso","Buonasera, sto riscontrando problemi con il modulo di acce
"Richiesta Verifica Dati Anagrafici","Salve, ho notato che i miei dati anagrafici p
"Errore Caricamento Materiali","Ciao, durante la preparazione delle lezioni ho cerc
"Aggiornamento Voti Scrutinio","Gentile assistenza, come professore coordinatore ho
```

Figura 2.4: Risposta corretta

Da questo punto in poi il modello ha continuato la generazione del dataset rispondendo ai prompt fornitogli. La conversazione è continuata con alcuni problemi come mostrato dalla Figura 2.4 dove nonostante la richiesta esplicita di più record il modello rispondesse con sole 20 righe. Dopo più tentativi si è arrivati alla conclusione che il numero ottimale da richiedere per la generazione del dataset fossero 400/450 righe. Anche se questo valore forniva spesso output accettabili, non era esente da problematiche come è possibile notare dalla Figura 2.5.

Impossibile visualizzare questo messaggio a causa di un errore.

Figura 2.5: Errore nella generazione

Sin dalle prime generazioni, è emerso chiaramente come l'uso di un dataset sintetico potesse influenzare significativamente le prestazioni del modello. Le frasi generate risultavano spesso ripetitive o poco adatte al contesto, evidenziando una marcata distanza da un dataset realistico. Per mitigare queste limitazioni, sono stati sperimentati diversi approcci, tra cui l'introduzione di richieste più creative e l'adozione di stili diversi in linea con il contesto di una scuola secondaria. Tuttavia, i risultati ottenuti non sono stati del tutto soddisfacenti. Alcune migliorie sarebbero potute essere quelle di aggiungere altre caretteristiche al dataset, ad esempio da che tipo di utente provenisse il messaggio (Docente o Genitore).

Capitolo 3

Text Extraction e Cleanup

3.1 Introduzione

La fase di **Text Extraction e Cleanup** si occupa dell'estrazione dei dati testuali dalle fonti e della successiva normalizzazione e pulizia, rendendo il testo più coerente e privo di rumore. Queste operazioni costituiscono il primo passo fondamentale per garantire che il testo sia pronto per l'analisi e per le ulteriori operazioni di pre-processing.

3.2 Caricamento e Fusione dei Dataset

I dati sono letti da file CSV. Utilizzando la libreria `pandas`, due dataset, ad esempio D_1 e D_2, vengono fusi per formare un nuovo dataset D. Ogni elemento del dataset risultante è definito come:

$$d = (\texttt{titolo\_messaggio}, \texttt{categoria}),$$

con

$$\texttt{titolo\_messaggio} = \texttt{titolo} \oplus \texttt{" "} \oplus \texttt{messaggio},$$

dove $\oplus$ rappresenta la concatenazione di stringhe. La seguente funzione implementa tale operazione:

Listato 3.1: Funzione per la fusione dei DataFrame

```
def merge_dataframes(frame1: pd.DataFrame, frame2: pd.DataFrame)
   -> pd.DataFrame:
    """
    Merge two dataframes and combine 'titolo' and 'messaggio'
       into a single column.
    """
    logger.info("Merging␣dataframes...")
    frame = pd.concat([frame1, frame2])
    num_duplicated = frame.duplicated().sum()
    frame.drop_duplicates(inplace=True)
    logger.info("Eliminated␣%s␣duplicate␣rows", num_duplicated)
    frame['titolo_messaggio'] = frame['titolo'] + '␣' + frame['
       messaggio']
    return frame[['titolo_messaggio', 'categoria']]
```

3.3 Operazioni di Pulizia e Normalizzazione

Questa fase consiste nel ridurre la complessità del testo, eliminando elementi non informativi e standardizzando la rappresentazione.

3.3.1 Conversione in Minuscolo e Rimozione degli Spazi Inutili

Per eliminare la distinzione tra maiuscole e minuscole e rimuovere spazi in eccesso, il testo viene convertito in minuscolo e sottoposto a stripping. Formalmente, per una stringa s:

$$s' = \text{lowercase}(s)$$

seguendo la rimozione di spazi iniziali e finali, oltre alla compressione di spazi multipli.

3.3.2 Rimozione di URL, Punteggiatura e Numeri

Utilizzando espressioni regolari, si eliminano pattern specifici come URL (corrispondenti a `https?://\S+` o `www\.\S+`), punteggiatura e numeri, che possono introdurre rumore nella rappresentazione del testo.

3.3.3 Eliminazione di Saluti e Formule di Cortesia

Le formule di saluto (es. "ciao", "buongiorno") vengono identificate e rimosse dalla text body per focalizzare l'analisi sui contenuti informativi.

3.3.4 Tokenizzazione e Correzione Ortografica

La tokenizzazione segmenta il testo in parole (token) e, contemporaneamente, una correzione ortografica viene applicata per uniformare termini scritti in maniera incoerente. La funzione seguente realizza l'intero processo di pulizia e normalizzazione:

Listato 3.2: Funzione minimal_preprocess

```
def minimal_preprocess(text: str) -> str:
    text = text.lower().strip()
    text = re.sub(r'https?://\S+|www\.\S+', '', text)
    text = remove_greetings_secretary(text)
    text = re.sub(r'[^\w\s]', '␣', text)
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'\s+', '␣', text)
    tokens = text.split()
    corrected_tokens = []
    for word in tokens:
        corrected_word = spell.correction(word)
        corrected_tokens.append(corrected_word if corrected_word
            else word)
    return '␣'.join(corrected_tokens).strip()
```

Quest'ultima operazione insieme alle precedenti sono essenziali per preparare dati testuali grezzi e renderli utili per analisi successive. Attraverso la fusione dei dataset e una rigorosa pulizia del testo, si ottiene una rappresentazione priva di rumore e pronta per le fasi di pre-processing.

Capitolo 4

Pre Processing

4.1 Introduzione

La fase di pre-processing approfondisce la trasformazione del testo già estratto e pulito, arricchendo la rappresentazione semantica mediante operazioni quali tokenizzazione, lemmatizzazione e Named Entity Recognition (NER). Queste tecniche avanzate migliorano l'analisi computazionale e preparano il testo per algoritmi di machine learning.

4.2 Tokenizzazione, Lemmatizzazione e Named Entity Recognition

Per incrementare il valore semantico del testo, vengono applicate le seguenti operazioni:

- **Tokenizzazione:** segmenta il testo in unità minime (token).
- **Lemmatizzazione:** riduce ogni token alla sua forma base, diminuendo la variabilità morfologica.
- **Named Entity Recognition (NER):** identifica entità come nomi, organizzazioni e località, etichettandole come ad esempio `NER_PERSON`.

La seguente funzione integra questi passaggi:

Listato 4.1: Funzione per il processing del testo

```
def process_text(text):
    cleaned_text = minimal_preprocess(text)
    doc = nlp(cleaned_text)
    tokens = []
    for token in doc:
        if token.is_stop or token.is_punct or token.is_space:
            continue
        lemma = token.lemma_.strip()
        if lemma:
            tokens.append(lemma)
    for ent in doc.ents:
        tokens.append("NER_%s" % ent.label_)
    return ' '.join(tokens)
```

4.2.1 Deep Learning in spaCy per il Riconoscimento delle Entità

spaCy sfrutta modelli di deep learning che combinano:

- **Reti Neurali Convoluzionali (CNN):** per estrarre informazioni locali dal testo.
- **Meccanismi di Attenzione:** per comprendere il contesto attorno a ciascun token.
- **Architetture Trasformative:** per individuare relazioni tra token anche se distanti nel testo.

Queste tecnologie consentono di raggiungere un elevato livello di accuratezza nel riconoscimento delle entità, anche in testi complessi.

4.3 Parallelizzazione del Pre-processing

Per gestire dataset di grandi dimensioni, la parallelizzazione sfrutta la libreria `joblib` per distribuire il processo su più core. Il backend "threading" viene utilizzato per ottimizzare l'uso delle risorse hardware. Esempio:

Listato 4.2: Funzione parallel_process_texts

```
def parallel_process_texts(series, n_jobs=-1):
    logger.info("Parallel text processing with threading backend ...")
    with parallel_backend('threading', n_jobs=n_jobs):
        processed = Parallel()(delayed(process_text)(text) for text in series)
    return pd.Series(processed, index=series.index)
```

Capitolo 5

Feature Engineering

5.1 Introduzione

La fase di Feature Engineering è cruciale nel processo di sviluppo di modelli di machine learning, specialmente in ambito Natural Language Processing (NLP). Questa fase si occupa della trasformazione dei dati grezzi in rappresentazioni numeriche che catturino le caratteristiche essenziali dei testi. Un'accurata progettazione in questa fase riduce il rumore e la complessità computazionale, ponendo solide basi per le successive attività di modellazione.

5.2 Trasformazione del Testo con TF-IDF

L'approccio TF-IDF (Term Frequency-Inverse Document Frequency) è una tecnica consolidata per convertire testi in vettori numerici. Essa assegna un peso a ciascun termine in un documento, rendendo più rilevanti i termini discriminanti rispetto a quelli troppo comuni. Formalmente, il peso $w(t, d)$ per un termine t in un documento d appartenente ad un corpus D è definito da:

$$w(t, d) = \mathrm{tf}(t, d) \cdot \log \frac{N}{\mathrm{df}(t)}$$

Nella quale:

- $\mathrm{tf}(t, d)$ è il conteggio del termine t nel documento d;
- N rappresenta il numero totale dei documenti nel corpus;
- $\mathrm{df}(t)$ indica il numero di documenti in cui il termine t appare.

I parametri comuni del vettorizzatore TF-IDF sono:

- `use_idf=True`: attiva la ponderazione inversa.
- `ngram_range=(1, 2)`: include sia unigrammi che bigrammi per cogliere relazioni tra parole.
- `max_features=3000`: limita il vocabolario ai termini più rilevanti, migliorando l'efficienza.

- `norm='l2'`: applica la normalizzazione per uniformare i vettori.
- `smooth_idf=True`: evita problemi di divisione per zero mediante una regolarizzazione dell'IDF.
- `sublinear_tf=True`: applica una scala logaritmica per attenuare l'effetto di termini dalle frequenze molto elevate.

5.3 Confronto tra Word2Vec e TF-IDF

5.3.1 Word2Vec: Definizione e Caratteristiche

Definizione 5.1 (Word2Vec). *Word2Vec è un insieme di modelli di rete neurale che producono word embedding, ovvero rappresentazioni vettoriali distribuite di parole in uno spazio continuo a n dimensioni. Formalmente, dato un vocabolario V, Word2Vec apprende una funzione $\phi : V \to \mathbb{R}^n$ che mappa ogni parola $w \in V$ in un vettore $\vec{w} \in \mathbb{R}^n$, dove n è la dimensione dello spazio di embedding.*

Word2Vec utilizza due architetture principali:

- **Continuous Bag of Words (CBOW):** predice una parola target date le parole del contesto. La probabilità condizionata è definita come:

$$P(w_t|w_{t-k}, ..., w_{t-1}, w_{t+1}, ..., w_{t+k}) = \frac{\exp(\vec{w_t}^T \cdot \vec{h})}{\sum_{w' \in V} \exp(\vec{w'}^T \cdot \vec{h})}$$

  dove $\vec{h}$ è la media dei vettori delle parole di contesto.

- **Skip-gram:** predice le parole del contesto data una parola target.

5.3.2 Motivazione della Scelta di TF-IDF

La scelta di utilizzare TF-IDF invece di Word2Vec è stata guidata da diverse considerazioni teoriche e pratiche:

1. **Interpretabilità:** TF-IDF produce rappresentazioni sparse e direttamente interpretabili, dove ogni dimensione corrisponde a una specifica parola del vocabolario.
2. **Efficienza Computazionale:** TF-IDF non richiede una fase di pre-training su grandi corpora di testo, risultando computazionalmente più efficiente. La complessità temporale per la costruzione della matrice TF-IDF è $O(|D| \cdot \bar{L})$, dove $|D|$ è il numero di documenti e $\bar{L}$ è la lunghezza media dei documenti.
3. **Dimensione del Dataset:** Word2Vec richiede tipicamente grandi quantità di dati per apprendere embedding significativi. Per un dataset di dimensioni moderate, TF-IDF offre una rappresentazione più robusta senza il rischio di underfitting.
4. **Specificità del Dominio:** TF-IDF cattura efficacemente l'importanza relativa dei termini nel contesto specifico del corpus, mentre Word2Vec potrebbe introdurre bias dovuti al pre-training su domini diversi.

5.4 Riduzione della Dimensionalità tramite Truncated SVD

La rappresentazione TF-IDF genera uno spazio vettoriale di elevata dimensionalità, spesso sparso. La tecnica di Truncated Singular Value Decomposition (SVD) riduce la dimensionalità mantenendo la maggior parte della varianza informativa. Data una matrice $A \in \mathbb{R}^{m \times n}$ ottenuta dal TF-IDF, la SVD decompone la matrice nel seguente modo:

$$A = U\Sigma V^T,$$

Nella quale:

- U e V sono matrici ortogonali;
- Σ è una matrice diagonale con i valori singolari in ordine decrescente.

Aplicando la Truncated SVD, si conserva un numero ridotto k di valori singolari:

$$A_k = U_k \Sigma_k V_k^T,$$

con k scelto in base all'analisi della varianza da preservare (ad esempio, 30, 50, 100). Questa operazione consente di semplificare il modello, eliminando rumore e ridondanze nel set di feature.

La riduzione della dimensionalità tramite Truncated SV insieme alla trasformazione del testo con TF-IDF permettono di ottenere una rappresentazione dei dati testuali efficace e compatta. Tali tecniche , migliorano le prestazioni computazionali e la generalizzazione del modello di machine learning.

Capitolo 6

Model Building

6.1 Introduzione

Il processo di *Model Building* è fondamentale per tradurre le rappresentazioni numeriche ottenute in fase di Feature Engineering in un modello capace di classificare testi in modo accurato e robusto. In questo capitolo si descrive formalmente la composizione di una pipeline modulare, l'ottimizzazione degli iperparametri e la scelta del classificatore, fornendo definizioni matematiche ed esempi di implementazione.

6.2 Definizione della Pipeline

Definizione 6.1 (Pipeline di Machine Learning). *Una pipeline di machine learning è definita come una sequenza ordinata di trasformazioni dati, implementata per automatizzare il flusso di lavoro di modellazione.* Formalmente, una pipeline $\mathcal{P}$ è una funzione composita di n trasformazioni f_i:

$$\mathcal{P}(x) = f_n \circ f_{n-1} \circ \cdots \circ f_1(x),$$

dove ciascun f_i rappresenta una trasformazione applicata al dato grezzo x. L'output finale $\mathcal{P}(x)$ è la predizione del modello, ossia l'etichetta di classe assegnata. La pipeline si compone dei seguenti elementi:

1. **Vettorizzazione:** Converte il testo in vettori numerici mediante TF-IDF.
2. **(Opzionale) Riduzione della Dimensionalità:** Utilizza tecniche come la Truncated SVD per ridurre lo spazio delle feature, soprattutto se il modello è molto complesso.
3. **Classificazione:** Addestra un classificatore, ad esempio *Naive Bayes* o *Support Vector Machine (SVM)*, per assegnare ad ogni documento un'etichetta.

6.3 Scelta del Classificatore

La scelta del classificatore dipende dalla natura dei dati e dalle esigenze del problema:

- **Naive Bayes:** basato su un modello probabilistico che assume l'indipendenza condizionale delle feature. È particolarmente efficace per dati testuali e regola il proprio comportamento tramite il parametro `alpha`.

  Definizione 6.2 (Classificatore Naive Bayes). *Un classificatore Naive Bayes è un algoritmo di classificazione probabilistico basato sull'applicazione del teorema di Bayes con l'assunzione "naive" di indipendenza condizionale tra ogni coppia di feature data la classe della variabile.*

  Per il testo, questa assunzione si traduce nell'ipotesi che la presenza di una parola in un documento sia indipendente dalla presenza di altre parole, dato l'argomento del documento. Il parametro `alpha` in `MultinomialNB` è un parametro di smoothing additivo, che previene probabilità nulle per parole non viste nei dati di training.

- **Support Vector Machine (SVM):** utilizza un kernel lineare per individuare l'iperpiano che separa al meglio le diverse classi. L'opzione `probability=True` permette di stimare probabilità, mentre il parametro `C` gestisce il compromesso tra margine e errore di classificazione.

  Definizione 6.3 (Support Vector Machine - SVM). *Un Support Vector Machine (SVM) è un algoritmo di apprendimento supervisionato utilizzato per la classificazione e la regressione. In un contesto di classificazione binaria, un SVM mira a trovare l'iperpiano ottimale che massimizza il margine tra le due classi nello spazio delle feature. Per dati non linearmente separabili, SVM utilizza funzioni kernel per mappare i dati in uno spazio di dimensionalità superiore dove è possibile trovare un iperpiano lineare.*

  Il parametro `C` è un parametro di regolarizzazione che controlla il trade-off tra massimizzare il margine e minimizzare l'errore di classificazione sui dati di training. Un valore minore di `C` privilegia un margine più ampio, potenzialmente portando a una maggiore generalizzazione, mentre un valore maggiore di `C` cerca di classificare correttamente tutti i punti di training, rischiando l'overfitting. L'opzione `probability=True` abilita la stima della probabilità di appartenenza alla classe, utilizzando la cross-validazione.

6.4 Implementazione della Pipeline

Lo snippet di codice seguente (`pipeline.py`) illustra la costruzione della pipeline in Python. Tale esempio mostra la configurazione del vettorizzatore TF-IDF, la possibile applicazione della Truncated SVD e la scelta del classificatore.

Listato 6.1: File pipeline.py

```
from typing import Tuple, Dict, Any
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC

class ClassifierType:
    NAIVE_BAYES = 'naive_bayes'
    SVM = 'svm'

def build_pipeline(classifier_type: str) -> Tuple[Pipeline, Dict
   [str, Any]]:
    # Configurazione del vettorizzatore TF-IDF
    tfidf = TfidfVectorizer(
        use_idf=True,
        ngram_range=(1, 2),
        max_features=3000,
        norm='l2',
        smooth_idf=True,
        sublinear_tf=True
    )

    if classifier_type == ClassifierType.NAIVE_BAYES:
        classifier = MultinomialNB()
        pipeline = Pipeline([
            ('tfidf', tfidf),
            ('clf', classifier)
        ])
        param_grid = {
            'tfidf__min_df': [1, 3],
            'tfidf__max_df': [0.85, 0.90],
            'clf__alpha': [1.0, 1.5, 2.0]
        }
    elif classifier_type == ClassifierType.SVM:
        # In SVM, si applica la Truncated SVD per ridurre la
           dimensionalita
        svd = TruncatedSVD(n_components=100, random_state=42)
        classifier = SVC(probability=True, kernel='linear',
           random_state=42)
        pipeline = Pipeline([
            ('tfidf', tfidf),
            ('svd', svd),
            ('clf', classifier)
```

```
        ])
        param_grid = {
            'tfidf__min_df': [1, 3],
            'tfidf__max_df': [0.85, 0.90],
            'svd__n_components': [30, 50, 100, 150],
            'clf__C': [0.1, 0.5, 1.0, 2.0]
        }
    else:
        raise ValueError("Classifier type not supported.")

    return pipeline, param_grid
```

6.5 Ottimizzazione degli Iperparametri

L'ottimizzazione degli iperparametri viene realizzata tramite Grid Search. Formalmente, si cerca:

$$\theta^* = \arg\min_{\theta \in \Theta} J_{\text{CV}}(\theta),$$

dove $J_{\text{CV}}(\theta)$ è il costo medio stimato mediante validazione incrociata (ad esempio, con 5 fold).

Definizione 6.4 (Grid Search). *Grid Search è un algoritmo di ottimizzazione degli iperparametri che esegue una ricerca esaustiva attraverso un sottoinsieme definito dallo spazio degli iperparametri di un modello. Per ogni combinazione di iperparametri nella griglia, valuta le prestazioni del modello utilizzando la cross-validazione. La combinazione che produce le migliori prestazioni (secondo una metrica di valutazione scelta, come l'accuratezza o l'F1-score) viene selezionata come la configurazione ottimale degli iperparametri.*

Il seguente snippet mostra un esempio dell'utilizzo di Grid Search per trovare la migliore combinazione di iperparametri.

```
from typing import Tuple, Dict, Any, Optional
import pandas as pd
from sklearn.model_selection import KFold, GridSearchCV
from build_pipeline import build_pipeline, ClassifierType

def run_grid_search(x_data: pd.Series,
                    y_data: pd.Series,
                    classifier_type: str,
                    monitor: bool = False
                   ) -> Tuple[GridSearchCV, Optional[Dict[str,
                     Any]]]:
    pipeline, param_grid = build_pipeline(classifier_type)
    # Suddivisione dei dati in 5 fold per la validazione
       incrociata
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    grid_search = GridSearchCV(pipeline, param_grid, cv=kf,
       n_jobs=-1, verbose=0)
```

```
18      if monitor:
19          # Possibile implementazione del monitoraggio delle
               risorse (CPU, memoria, tempo)
20          pass
21      else:
22          grid_search.fit(x_data, y_data)
23
24      return grid_search, None
```

6.6 Funzioni di Inferenza: `get_model` e `predict_category`

Questa sezione descrive in dettaglio le funzioni `get_model` e `predict_category`, cruciali per la fase di inferenza del modello. Queste funzioni incapsulano rispettivamente il caricamento e l'addestramento del modello, e la predizione della categoria di nuovi messaggi.

6.6.1 `get_model`

La funzione `get_model(classifier_type: ClassifierType) -> Pipeline` gestisce il caricamento e l'addestramento del modello di classificazione.
Definizione 6.5 (Modello Pre-addestrato). *un modello pre-addestrato è un modello di machine learning che è stato precedentemente addestrato su un dataset di grandi dimensioni.*
In questo contesto, un modello pre-addestrato viene salvato su disco dopo la fase di ottimizzazione degli iperparametri per essere riutilizzato senza dover ripetere l'intero processo di addestramento. La funzione opera come segue:

1. **Caricamento del Modello Pre-addestrato:** tenta di caricare un modello pre-addestrato dal disco per il `classifier_type` specificato.

2. **Addestramento tramite Grid Search (se necessario):** se non viene trovato alcun modello salvato, la funzione procede con l'addestramento di un nuovo modello, utilizzando i dati di training $X_{\text{processed}}$ e le etichette y.

3. **Salvataggio del Modello:** dopo l'addestramento, il modello ottimizzato viene salvato su disco per utilizzi futuri.

4. **Restituzione del Modello:** la funzione restituisce il modello, sia esso caricato o appena addestrato.

Listato 6.2: Funzione get_model

```
def get_model(classifier_type: ClassifierType) -> Pipeline:
    """
    Load a pre-trained model if available; otherwise, train via
        grid search and save the model.
    Returns the model.
    """
    model = load_model(classifier_type)
    if model is None:
        logger.info("No saved model found for %s. Training a
            new one...", classifier_type.value)
        model = perform_grid_search(X_processed, y,
            classifier_type)
        save_model(model, classifier_type)
    else:
        logger.info("Using saved model for %s.",
            classifier_type.value)
    return model
```

6.6.2 predict_category

La funzione `predict_category(message: str, classifier_type: ClassifierType, top_n: int = 3) -> List[Tuple[Any, float]]` è responsabile della predizione della categoria per un nuovo messaggio di testo.

Definizione 6.6 (Predizione di Categoria). *La predizione di categoria è il processo di assegnazione di una o più etichette di categoria predefinite a un nuovo documento di testo, basandosi su un modello di classificazione addestrato.*

La funzione esegue i seguenti passi:

1. **Pre-processamento del Messaggio di Input:** il messaggio di input viene pre-processato utilizzando la funzione `process_text`, che include le operazioni di pulizia e normalizzazione del testo descritte nel capitolo precedente.

2. **Predizione e Probabilità:** utilizza il modello caricato per predire la categoria del messaggio pre-processato. Se il modello supporta la stima delle probabilità (come nel caso di SVM con `probability=True` e Naive Bayes), la funzione calcola le probabilità per ciascuna classe.

3. **Restituzione delle Top-N Predizioni:** se sono disponibili le probabilità, la funzione restituisce una lista delle `top_n` categorie più probabili, ordinate per probabilità decrescente, insieme alle rispettive probabilità.

Listato 6.3: Funzione predict_category

```python
def predict_category(
        message: str,
        classifier_type: ClassifierType,
        top_n: int = 3
) -> List[Tuple[Any, float]]:
    """
    Preprocess the input message and predict its category using the specified classifier.
    Returns top-N predicted labels and probabilities if available.
    """
    logger.info("Predicting category for a new message...")
    model = get_model(classifier_type)
    processed_message = process_text(message)
    clf = model.named_steps.get('clf', model)  # Use dict-like access if available

    if hasattr(clf, "predict_proba"):
        probs = model.predict_proba([processed_message])[0]
        classes = clf.classes_
        sorted_indices = np.argsort(probs)[::-1]
        top_n = min(top_n, len(classes))
        predictions = [(classes[i], probs[i]) for i in sorted_indices[:top_n]]
        return predictions

    # Fallback: predict single label with probability 1.0
    category = model.predict([processed_message])[0]
    return [(category, 1.0)]
```

Capitolo 7

Evaluation

7.1 Introduzione

In questo capitolo, sono esaminati i risultati ottenuti dalla **validazione incrociata** (**K-fold cross-validation**) sui modelli di classificazione **Naive Bayes** e **Support Vector Machine** (**SVM**). L'obiettivo principale è valutare le prestazioni di ciascun modello in modo sistematico e approfondito, utilizzando un approccio rigoroso basato su **metriche quantitative**.

Per ottenere un'analisi dettagliata, sono state misurate le performance di entrambi i modelli attraverso metriche chiave come **accuratezza**, **matrice di confusione**, **precisione**, **recall**, **F1-score** e **overfitting**. La **validazione incrociata** consente di ridurre la variabilità nei risultati e garantire che le **prestazioni osservate** non dipendano da una particolare suddivisione dei dati. In questo contesto, il **confronto** tra Naive Bayes e SVM permette di evidenziare differenze significative in termini di **capacità predittiva**, **robustezza** e **adattabilità** a differenti **distribuzioni dei dati**.

Infine, sono discussi dei possibili **scenari applicativi** in cui ciascun modello potrebbe risultare più adatto, considerando fattori come la **sensibilità ai dati sbilanciati**, le **risorse computazionali richieste** e l'**interpretabilità del modello**.

7.2 Metriche di Valutazione dei Modelli

Per valutare le prestazioni dei modelli, sono state utilizzate le seguenti metriche:

- **Accuratezza**: misura la percentuale di previsioni corrette rispetto al numero totale di previsioni.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

- **Precisione** (o **Positive Predictive Value**): indica la percentuale di previsioni positive corrette rispetto a tutte le previsioni positive fatte dal modello.

$$Precision = \frac{TP}{TP + FP} \tag{7.2}$$

- **Recall** (o **Sensibilità** o **True Positive Rate**): misura la percentuale di veri positivi identificati correttamente dal modello.

$$Recall = \frac{TP}{TP + FN} \tag{7.3}$$

- **F1-score**: è la media armonica di precisione e recall, che bilancia entrambi i valori. È utile quando c'è uno squilibrio tra le classi.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{7.4}$$

- **Overfitting**: misura la differenza tra l'accuratezza sui dati di addestramento e l'accuratezza sui dati di test per valutare la capacità di generalizzazione del modello. Un valore elevato di questa differenza indica che il modello si adatta troppo ai dati di training e potrebbe non generalizzare bene su dati non visti.

$$Overfitting = Accuracy_{train} - Accuracy_{test} \tag{7.5}$$

Dove:

- **TP** è il numero di veri positivi,
- **TN** è il numero di veri negativi,
- **FP** è il numero di falsi positivi,
- **FN** è il numero di falsi negativi.

7.3 Risultati del Modello Naive Bayes

7.3.1 Risultati di Addestramento

Il modello **Naive Bayes** ha ottenuto un'accuratezza complessiva del **92.43%** sui dati di addestramento. Questa performance suggerisce una buona capacità del modello di generalizzare il comportamento delle classi, mantenendo un buon livello di prestazioni anche su dati non visti.

La matrice di confusione, che riporta il numero di previsioni corrette e errate per ciascuna classe, mostra i seguenti risultati:

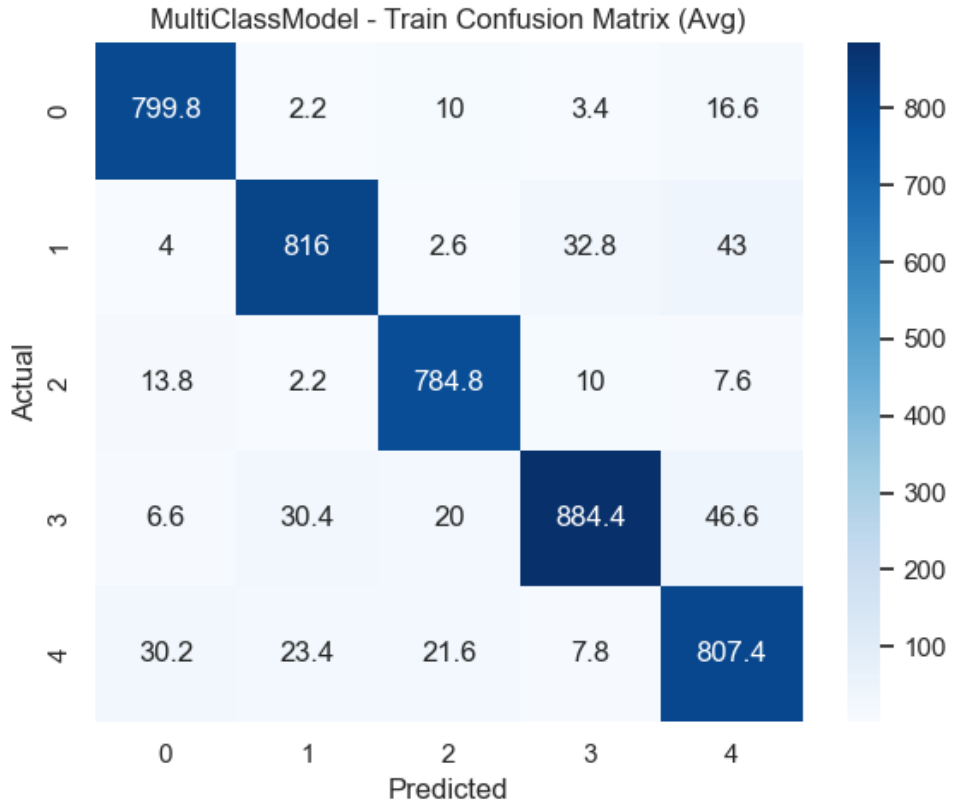

Figura 7.1: Matrice di confusione per il training di Naive Bayes

Il modello mostra buone prestazioni complessive, con un'alta percentuale di istanze correttamente classificate, come evidenziato dai valori elevati sulla **diagonale principale**. Tuttavia, ci sono errori significativi tra alcune classi, in particolare tra le classi 0, 1 e 4, dove il modello ha difficoltà a distinguere correttamente. Per esempio, molte istanze della classe 0 sono state erroneamente classificate come classe 4, e molte istanze della classe 1 sono state scambiate con le classi 3 e 4. Questo evidenzia alcune aree in cui il modello potrebbe migliorare nel riconoscere meglio le differenze tra queste classi.

Le metriche per ogni categoria sono mostrate nella seguente tabella e visualizzate nel seguente grafico:

Categoria	Precision	Recall	F1-score
Accesso	93.61%	96.13%	94.85%
Didattica	93.34%	90.82%	92.06%
Profilo	93.55%	95.89%	94.70%
Segreteria	94.24%	89.51%	91.82%
Tecnico	87.65%	90.67%	89.14%
Media	92.48%	92.61%	92.51%
Media pesata	92.49%	92.44%	92.43%

Tabella 7.1: Confronto tra precision, recall e F1-score per il training di Naive Bayes

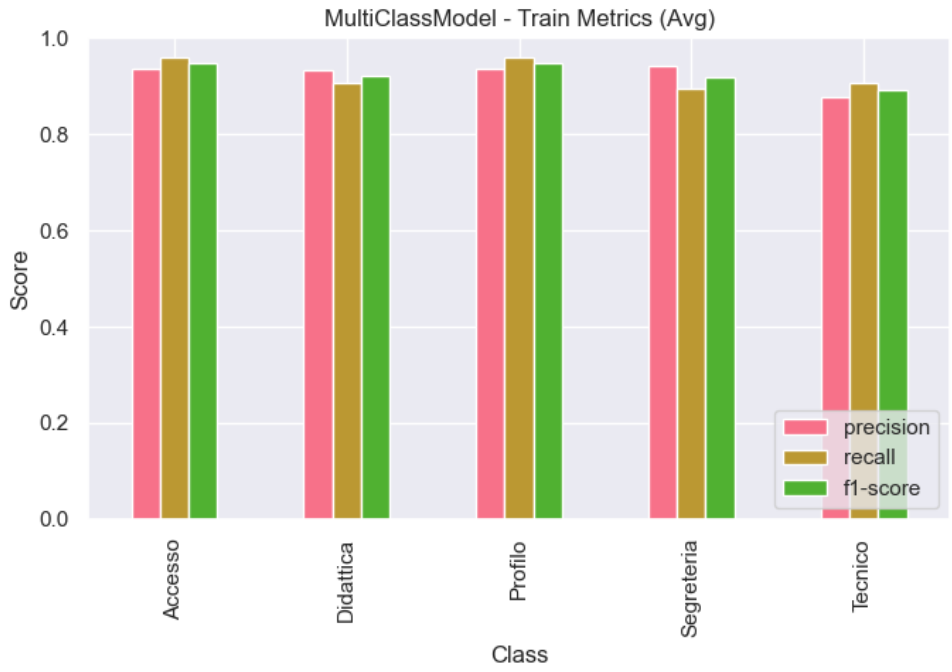

Figura 7.2: Confronto tra precision, recall e F1-score per il training di Naive Bayes

Il modello mostra buone performance complessive, ma nelle categorie **Segreteria** e **Tecnico** ci sono margini di miglioramento. **Segreteria** ha una buona precisione (94.24%), ma un recall relativamente basso (89.51%), mentre **Tecnico** ha un buon recall (90.67%), ma una precisione più bassa (87.65%).

7.3.2 Risultati di Test

Per quanto riguarda i **dati di test**, il modello ha ottenuto un'accuratezza del **88.92%**, una leggera riduzione rispetto ai dati di addestramento, ma comunque una performance robusta.

La matrice di confusione sui dati di test è la seguente:

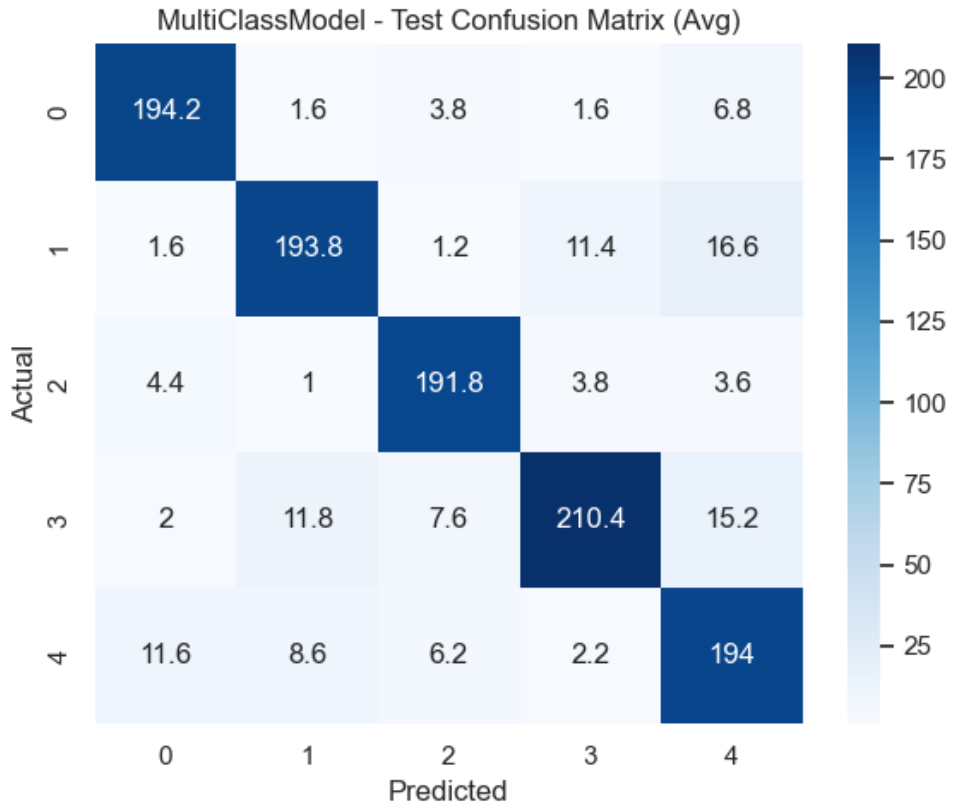

Figura 7.3: Matrice di confusione per il testing di Naive Bayes

Mentre le metriche per ogni categoria sono mostrate nella seguente tabella e visualizzate nel seguente grafico:

Categoria	**Precision**	**Recall**	**F1-score**
Accesso	90.83%	93.38%	92.07%
Didattica	89.35%	86.27%	87.77%
Profilo	91.02%	93.70%	92.33%
Segreteria	91.67%	85.15%	88.28%
Tecnico	82.18%	87.25%	84.63%
Media	89.01%	89.15%	89.02%
Media pesata	89.05%	88.92%	88.92%

Tabella 7.2: Confronto tra precision, recall e F1-score per il testing di Naive Bayes

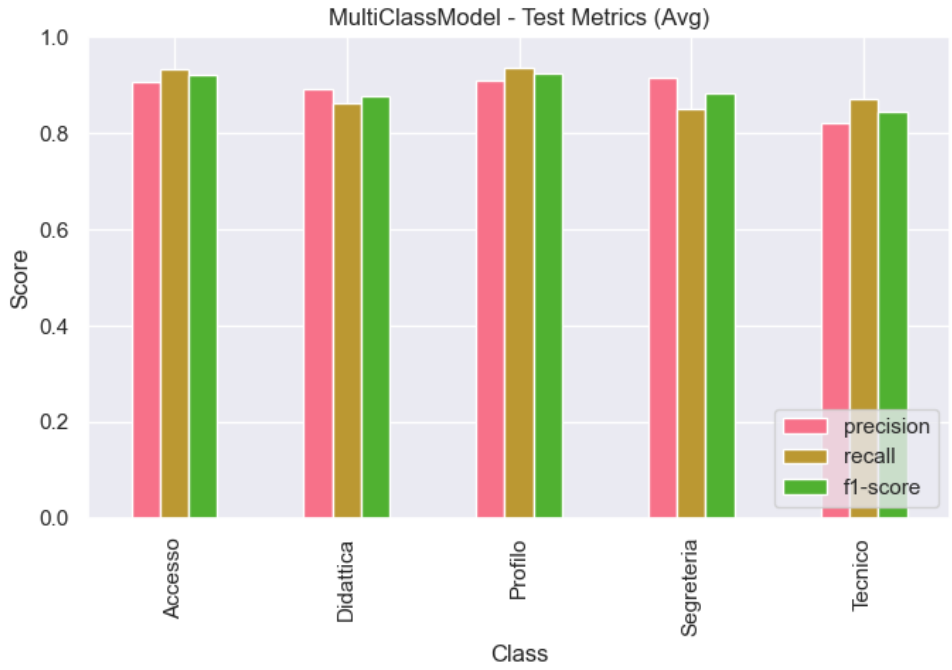

Figura 7.4: Confronto tra precision, recall e F1-score per il testing di Naive Bayes

7.3.3 Overfitting

Per valutare il rischio di **overfitting**, sono stati analizzati i valori di accuratezza nei diversi fold della validazione incrociata a k-fold. Le differenze tra l'accuratezza dei dati variano tra **0.02** e **0.05**, con una media di **0.0351**. Sebbene vi sia una leggera tendenza all'overfitting, i valori rientrano in un intervallo accettabile, confermando la buona generalizzazione del modello.

Figura 7.5: Confronto dell'overfitting nei 5 fold per Naive Bayes

7.4 Risultati del Modello Support Vector Machine (SVM)

7.4.1 Risultati di Addestramento

Il modello **Support Vector Machine (SVM)** ha ottenuto un'accuratezza complessiva del **92.06%** sui dati di addestramento. Questo risultato suggerisce una forte capacità del modello nel generalizzare il comportamento delle classi, mantenendo alte prestazioni anche su dati non visti.

La matrice di confusione, che riporta il numero di previsioni corrette ed errate per ciascuna classe, è la seguente:

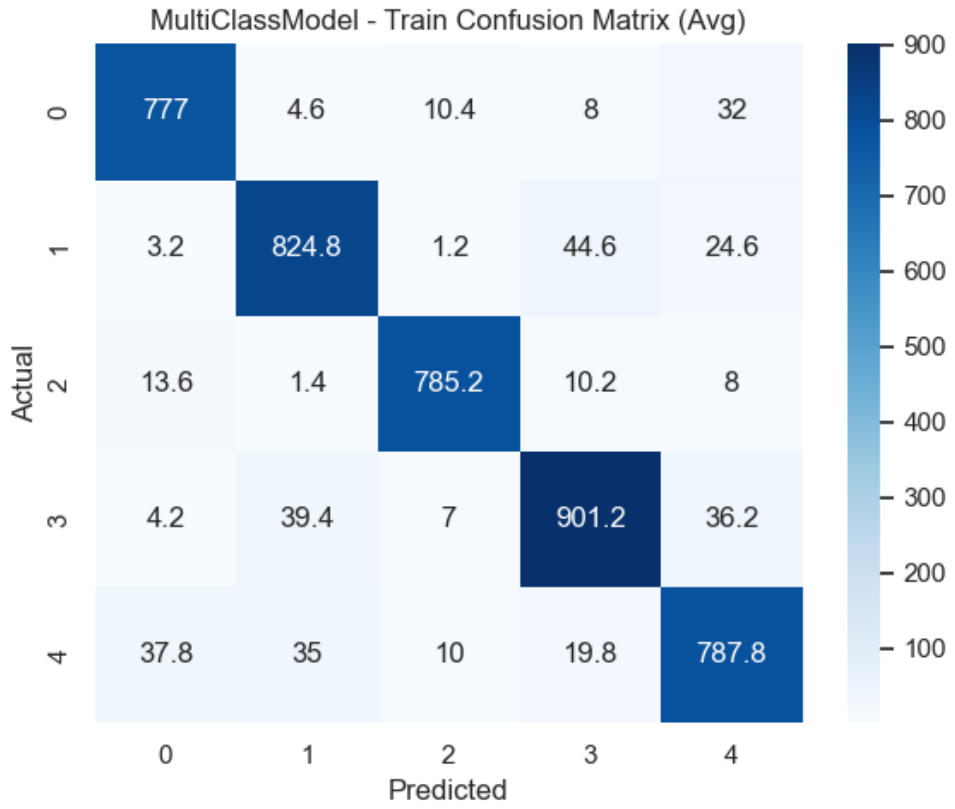

Figura 7.6: Matrice di confusione per i dati di addestramento con SVM

Il modello mostra buone prestazioni complessive, con un'alta percentuale di istanze correttamente classificate, come evidenziato dai valori elevati sulla **diagonale principale**. Tuttavia, ci sono errori significativi tra alcune classi, in particolare tra le classi 0, 3 e 4, dove il modello ha difficoltà a distinguere correttamente. Per esempio, molte istanze della classe 0 sono state erroneamente classificate come classe 4, mentre la classe 3 ha causato scambi con le classi 1 e 4. Inoltre, la classe 1 ha mostrato confusione con la classe 3, con molte istanze della classe 1 erroneamente classificate come classe 3. Questo evidenzia alcune aree in cui il modello potrebbe migliorare nel riconoscere meglio le differenze tra queste classi .

Le metriche per ogni categoria sono mostrate nella seguente tabella e visualizzate nel seguente grafico:

Categoria	Precision	Recall	F1-score
Accesso	92.97%	93.39%	93.18%
Didattica	91.13%	91.80%	91.46%
Profilo	96.49%	95.94%	96.21%
Segreteria	91.61%	91.21%	91.41%
Tecnico	88.66%	88.48%	88.57%
Media	92.17%	92.17%	92.16%
Media pesata	92.07%	92.07%	92.07%

Tabella 7.3: Confronto tra precision, recall e F1-score per il training di SVM

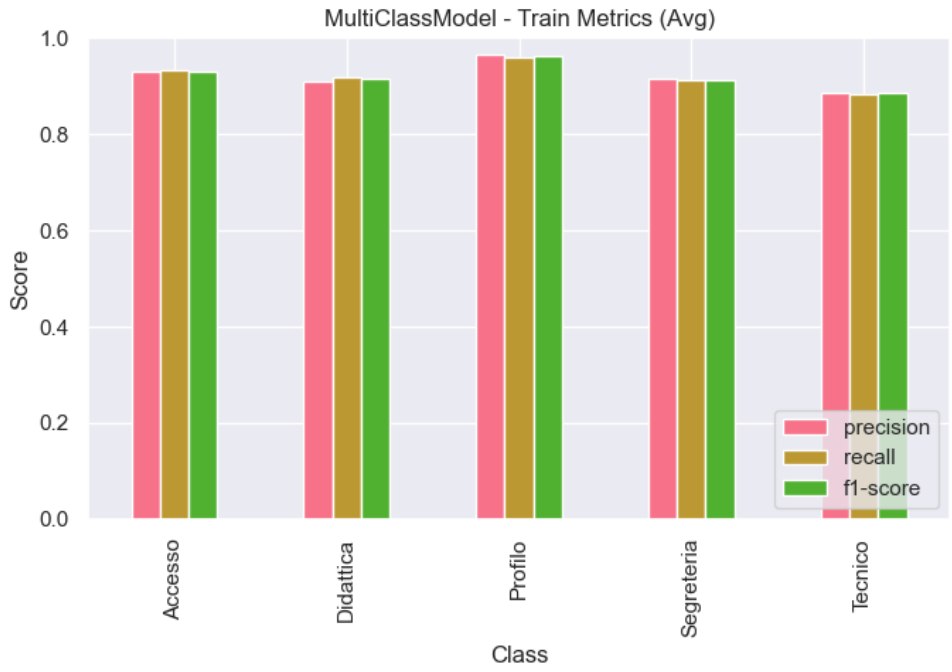

Figura 7.7: Confronto tra precision, recall e F1-score per il training di SVM

Il modello mostra buone performance complessive, ma nelle categorie **Tecnico** e **Didattica** ci sono margini di miglioramento. **Tecnico** ha una buona precisione (88.66%), ma un recall relativamente basso (88.48%), mentre **Didattica** ha un buon recall (91.80%), ma una precisione più bassa (91.13%).

7.4.2 Risultati di Test

Per quanto riguarda i **dati di test**, il modello ha ottenuto un'accuratezza del **89.95%**, con una leggera riduzione rispetto ai dati di addestramento, ma comunque una performance robusta.

La matrice di confusione sui dati di test è la seguente:

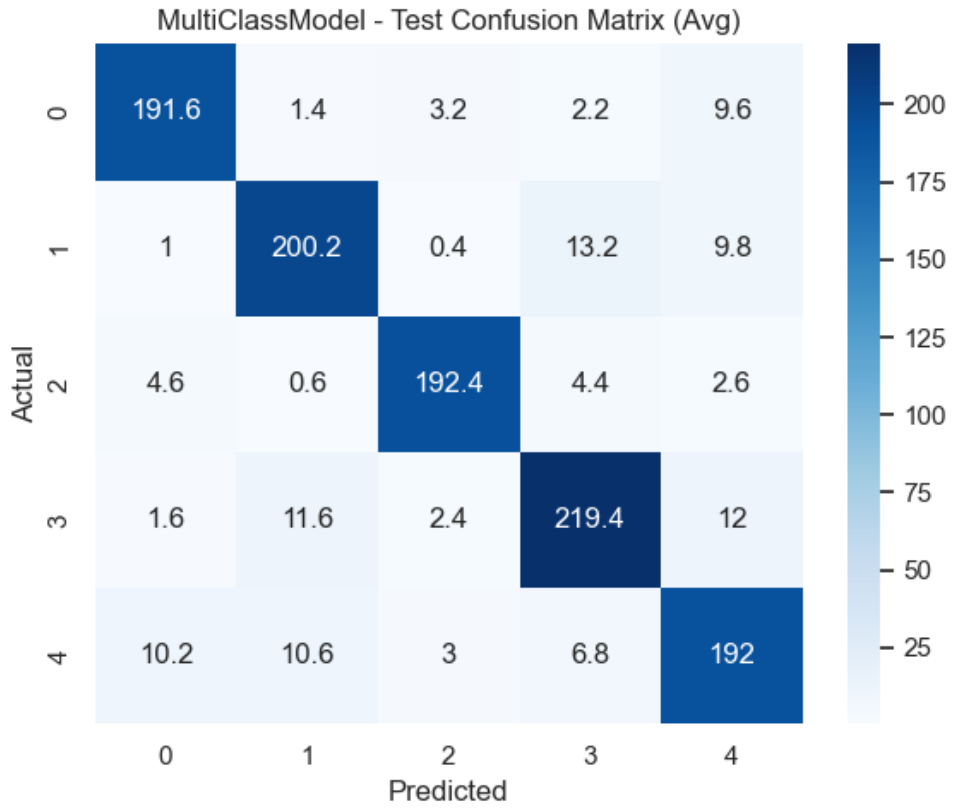

Figura 7.8: Matrice di confusione per i dati di test con SVM

Mentre le metriche per ogni categoria sono mostrate nella seguente tabella e visualizzate nel seguente grafico:

Categoria	Precision	Recall	F1-score
Accesso	91.67%	92.14%	91.89%
Didattica	89.19%	89.07%	89.13%
Profilo	95.50%	94.02%	94.74%
Segreteria	89.14%	88.81%	88.97%
Tecnico	85.14%	86.27%	85.69%
Media	90.13%	90.06%	90.08%
Media pesata	90.01%	89.95%	89.97%

Tabella 7.4: Confronto tra precision, recall e F1-score per il testing di SVM

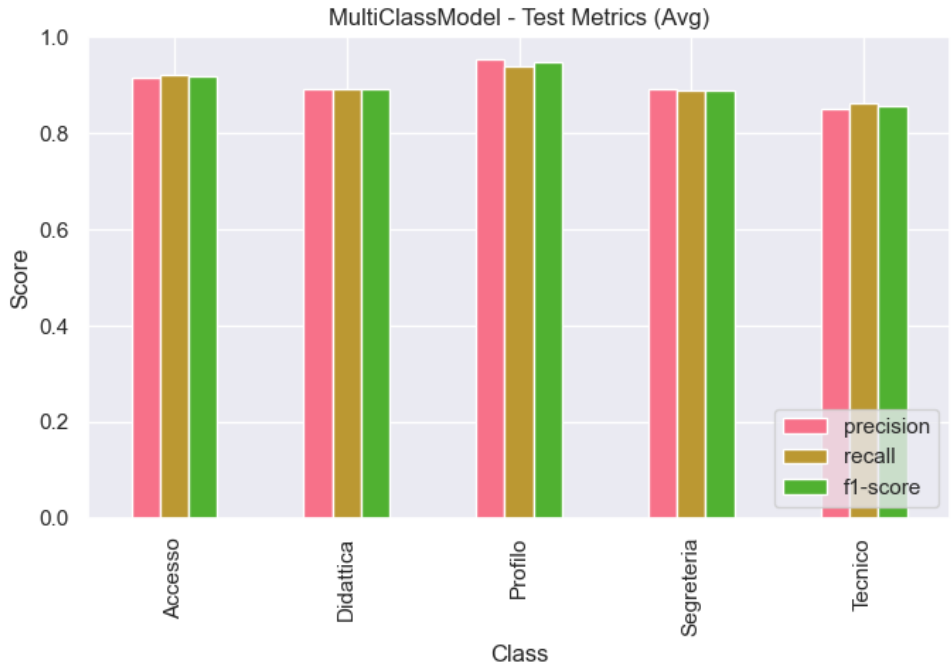

Figura 7.9: Confronto tra precision, recall e F1-score per il testing di SVM

7.4.3 Overfitting

Per valutare il rischio di **overfitting**, sono stati analizzati i valori di accuratezza nei diversi fold della validazione incrociata a k-fold. Le differenze tra l'accuratezza dei dati variano tra **0.01** e **0.04**, con una media di **0.0211**. Sebbene vi sia una leggera tendenza all'overfitting, i valori rientrano in un intervallo accettabile, confermando la buona generalizzazione del modello.

Figura 7.10: Confronto dell'overfitting nei 5 fold per SVM

7.5 Confronto tra i Modelli

7.5.1 Prestazioni Complessive

Entrambi i modelli, **Naive Bayes** e **SVM**, mostrano prestazioni molto simili. Durante il training, **Naive Bayes** raggiunge un'accuratezza del 92.44%, mentre **SVM** è leggermente migliore con un'accuratezza del 92.07%. Quando testati sui dati di test, entrambi i modelli registrano una piccola diminuzione delle performance, con **Naive Bayes** che scende al 88.92% e **SVM** che ottiene il 89.95%. Le metriche di precisione, recall e f1-score sono anch'esse molto simili per entrambi i modelli, suggerendo che entrambi sono abbastanza equilibrati nel bilanciare la capacità di classificare correttamente le istanze e nel ridurre al minimo gli errori.

Modello	Set	Accuracy	Precision	Recall	F1-score
Naive Bayes	Training	92.44%	92.49%	92.44%	92.43%
Naive Bayes	Test	88.92%	89.05%	88.92%	88.92%
SVM	Training	92.07%	92.17%	92.07%	92.07%
SVM	Test	89.95%	90.01%	89.95%	89.97%

Tabella 7.5: Confronto tra i modelli in termini di accuracy, precision, recall e F1-score

7.5.2 Overfitting

Sia il modello **Naive Bayes** che il modello **SVM** mostrano una lieve presenza di **overfitting**, con differenze tra le performance sui dati che variano tra **0.01** e **0.05** per Naive Bayes e tra **0.01** e **0.04** per SVM. Le differenze medie sono però significative: Naive Bayes ha una differenza media di **0.0351**, mentre SVM di **0.0211**, suggerendo che SVM ha una prestazione più stabile. L'andamento dell'overfitting è stato visualizzato calcolando le differenze per ogni fold del **k-fold cross-validation**, come mostrato nell'immagine seguente, che confronta i due modelli.

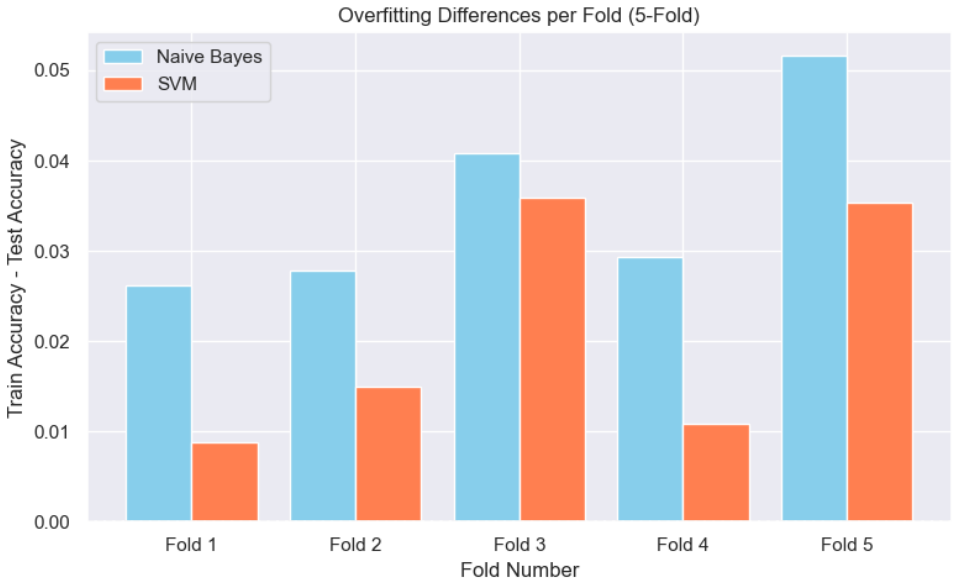

Figura 7.11: Confronto dell'overfitting tra Naive Bayes e SVM

7.6 Conclusioni

Entrambi i modelli, Naive Bayes e Support Vector Machine (SVM), hanno mostrato prestazioni eccellenti nel compito di **classificazione**, evidenziando differenze minime nelle metriche di valutazione. Sebbene il modello SVM ottenga risultati leggermente migliori in termini di accuratezza sui dati di test, con un'accuratezza del **89.95%** rispetto al **88.92%** di Naive Bayes, la differenza tra i due modelli è contenuta. Inoltre, i dati sull'overfitting suggeriscono che SVM presenta una prestazione più stabile, con una minore differenza media tra i fold della **validazione incrociata**, rispetto a Naive Bayes. Questo indica che entrambi i modelli sono ben calibrati per il compito e sono capaci di generalizzare efficacemente.

Naive Bayes, grazie alla sua semplicità e velocità di esecuzione, potrebbe essere la scelta preferibile in scenari con risorse computazionali limitate o in situazioni che richiedono modelli facilmente interpretabili. La sua capacità di gestire grandi volumi di dati con risorse relativamente basse lo rende adatto per applicazioni in tempo reale o su dispositivi con risorse limitate.

D'altra parte, **SVM** si distingue per una maggiore precisione in alcuni casi, ed è generalmente considerato una scelta solida quando è necessaria maggiore accuratezza e quando sono disponibili risorse computazionali adeguate. Inoltre, SVM è più adatto per situazioni che richiedono una maggiore capacità di generalizzazione.

In definitiva, entrambi i modelli sono validi per il problema in esame e presentano un buon compromesso tra **accuratezza** e capacità di **generalizzazione**. La scelta finale tra Naive Bayes e SVM dipenderà da considerazioni specifiche, come la disponibilità di risorse computazionali, la necessità di interpretabilità del modello e la priorità data a una leggera miglioria nelle prestazioni rispetto alla velocità di esecuzione.

Capitolo 8

Deployment

8.1 Server di Sviluppo Flask

Flask fornisce un server integrato che facilita il testing e lo sviluppo dell'applicazione. Il server viene configurato nel file principale dell'applicazione attraverso una semplice istruzione che ne definisce le modalità di esecuzione:

Listato 8.1: Configurazione Server Flask

```
if __name__ == '__main__':
    app.run(debug=True)
```

Definizione 8.1 (Server di Sviluppo) *Il server di sviluppo Flask è un web server leggero integrato nel framework, progettato per facilitare il processo di sviluppo e debug dell'applicazione.*

8.2 Endpoints dell'API

L'applicazione espone due endpoint principali per i servizi basati su intelligenza artificiale. Il primo endpoint gestisce l'orientamento degli studenti attraverso il modello Gemini di Google, mentre il secondo si occupa della classificazione dei ticket di supporto.

Definizione 8.2 (Endpoint) *Un endpoint rappresenta un punto di accesso specifico dell'API che risponde a determinate richieste HTTP. Nel contesto di questa applicazione, gli endpoint sono progettati per gestire richieste POST e restituire risposte in formato testuale con codifica UTF-8.*

8.3 Integrazione con Spring

L'applicazione Spring si interfaccia con il servizio Flask utilizzando RestTemplate per le comunicazioni HTTP. Questa integrazione permette di mantenere una separazione dei contesti applicativi garantendo al contempo una comunicazione efficiente tra i servizi.

Listato 8.2: Codice Integrazione Spring

```
private String callTicketService(String title, String message) {
    var restTemplate = new RestTemplate();
    var formData = new LinkedMultiValueMap<String, String>();
    formData.add("title", title);
    formData.add("message", message);

    var headers = new HttpHeaders();
    headers.setContentType(
        MediaType.APPLICATION_FORM_URLENCODED);

    var requestEntity = new HttpEntity<MultiValueMap<String,
       String>>(
        formData, headers);

    ResponseEntity<String> response = restTemplate.postForEntity(
        ticketServiceUrl,
        requestEntity,
        String.class);

    return response.getBody();
}
```

Definizione 8.3 (RestTemplate) *RestTemplate è una classe fornita da Spring Framework che semplifica l'interazione con servizi REST. Gestisce automaticamente la serializzazione e deserializzazione degli oggetti Java in richieste HTTP e viceversa.*

8.4 Configurazione

La configurazione dell'applicazione Flask include alcuni parametri fondamentali per il corretto funzionamento del servizio. In particolare, viene gestita la codifica JSON per supportare caratteri UTF-8 e viene configurato il sistema di logging per il monitoraggio dell'applicazione:

Listato 8.3: Configurazione base

```
app.config['JSON_AS_ASCII'] = False

logging.basicConfig(
    level=logging.INFO,
    format='[%(levelname)s] %(message)s'
)
```

Le variabili d'ambiente necessarie includono la chiave API per il servizio Gemini e l'URL del servizio ticket in Spring. Queste configurazioni devono essere gestite in modo appropriato nell'ambiente di deployment per garantire il corretto funzionamento dell'applicazione.

8.5 Installazione ed Esecuzione

Per procedere con l'installazione e l'esecuzione dell'applicazione *AstroMark AI*, si richiede di seguire attentamente i seguenti passaggi:

1. **Clonazione della Repository:**
   Clonare la repository ufficiale da GitHub eseguendo il comando:

   Listato 8.4: Clonazione della repository

```
git clone https://github.com/mariocosenza/astromark-ai.git
```

2. **Accesso alla Directory del Progetto:**
   Accedere alla cartella del progetto appena clonata:

   Listato 8.5: Accesso alla directory del progetto

```
cd astromark-ai
```

3. **Installazione delle Dipendenze:**
   Il file `requirements.txt` presente nella root del progetto elenca le librerie necessarie. Per installarle, eseguire:

   Listato 8.6: Installazione delle dipendenze

```
pip install -r requirements.txt
```

4. **Configurazione delle Variabili d'Ambiente:**
   Per abilitare l'integrazione con il servizio Gemini, è necessario impostare la variabile d'ambiente `GEMINI_API_KEY`. Tale operazione può essere effettuata secondo una delle seguenti modalità:

   a) **Creazione del file `.env`:**
   Nella directory principale del progetto, creare un file denominato `.env` contenente, ad esempio, la seguente istruzione:

   Listato 8.7: Esempio di configurazione nel file .env

```
GEMINI_API_KEY=your_gemini_api_key_here
```

   b) **Impostazione diretta tramite terminale:**
   In alternativa, è possibile definire la variabile d'ambiente direttamente tramite terminale eseguendo:

   Listato 8.8: Impostazione della variabile d'ambiente tramite terminale

```
export GEMINI_API_KEY=your_gemini_api_key_here
```

5. **Avvio dell'Applicazione:**
   Con Python 3.12 installato, avviare il server di sviluppo Flask eseguendo il file principale:

   Listato 8.9: Avvio del server Flask

```
python app.py
```

Capitolo 9

Conclusioni Finali

L'analisi e lo sviluppo di *AstroMark-AI* hanno evidenziato l'efficacia dell'approccio adottato per la classificazione testuale mediante tecniche di machine learning. Il confronto tra i modelli *Naive Bayes* e *Support Vector Machine (SVM)* ha mostrato prestazioni simili, con *SVM* che ha ottenuto un leggero vantaggio in termini di accuratezza, raggiungendo il 89.95% rispetto all'88.92% di *Naive Bayes*. Tuttavia, entrambi i modelli hanno dimostrato una buona generalizzazione e robustezza.

L'uso di *TF-IDF* come metodo di rappresentazione testuale si è rivelato efficace per l'elaborazione dei dati, garantendo un equilibrio tra interpretabilità e prestazioni computazionali. La pipeline di preprocessing e feature engineering ha permesso di migliorare la qualità dei dati, riducendo il rumore e ottimizzando la rappresentazione delle informazioni.

Dal punto di vista pratico, *Naive Bayes* si conferma una scelta ideale per applicazioni che richiedono rapidità di esecuzione e basso consumo di risorse, mentre *SVM* è preferibile nei casi in cui è necessaria una maggiore precisione. La decisione su quale modello adottare dipenderà quindi dal contesto applicativo e dai vincoli computazionali.

Infine, l'implementazione e il deployment del sistema su un'architettura basata su *Flask* e la sua integrazione con *Spring* hanno dimostrato la fattibilità di un sistema scalabile ed efficiente. Questo lavoro costituisce una base solida per futuri miglioramenti, tra cui l'esplorazione di modelli neurali più avanzati o l'integrazione di tecniche di apprendimento attivo per migliorare continuamente le prestazioni del sistema.

9.0.1 Limitazioni

Un aspetto critico di questo lavoro è la validazione delle performance utilizzando un dataset generato artificialmente tramite modelli di *Large Language Models* (LLM) piuttosto che dati raccolti da scenari d'uso reali. Questo introduce alcune limitazioni significative:

- **Mancanza di variabilità naturale**: i dati generati tendono a presentare una struttura e una distribuzione più regolari rispetto a quelli raccolti da utenti reali, riducendo la capacità del modello di adattarsi a espressioni linguistiche spontanee e meno formali.
- **Possibili bias nei dati sintetici**: poiché i dati sono generati da AI pre-addestrate su fonti specifiche, potrebbero riflettere le limitazioni intrinseche dei modelli linguistici, trasmettendo eventuali pregiudizi o schemi linguistici artificiali che non rispecchiano perfettamente l'uso effettivo del linguaggio.

- **Difficoltà nella generalizzazione**: un modello addestrato su dati sintetici potrebbe ottenere buone prestazioni sul dataset di test, ma riscontrare difficoltà significative quando applicato su dati provenienti da un ambiente reale, dove il linguaggio può essere più variegato e meno strutturato.
- **Assenza di rumore tipico delle interazioni reali**: nei dati reali sono presenti errori grammaticali, abbreviazioni, linguaggio informale e altre variabili che un dataset generato artificialmente non sempre cattura in modo fedele.

9.0.2 Lavori futuri

Per superare queste limitazioni e rendere il modello più robusto in contesti reali, alcune possibili direzioni future includono:

- L'adozione di modelli basati su reti neurali per migliorare la capacità di generalizzazione.
- L'integrazione di un sistema di *active learning* per affinare il modello con nuovi dati etichettati in modo incrementale.
- La raccolta e l'annotazione di dati reali per migliorare la qualità del dataset e confrontare le performance con dati sintetici.
- L'ottimizzazione delle prestazioni attraverso tecniche di compressione dei modelli e accelerazione hardware.

Con queste prospettive, *AstroMark-AI* si pone come una piattaforma versatile e in continua evoluzione, pronta a rispondere alle esigenze di analisi automatizzata del testo con strumenti di intelligenza artificiale.

Appendice A

Modelli Generativi e l'Orientamento Universitario

Premessa

La presente appendice approfondisce due tematiche di rilevanza nel contesto educativo: l'impiego di modelli generativi e la problematica dell'orientamento universitario. Quest'ultimo aspetto è particolarmente cruciale per gli studenti di scuola secondaria di secondo grado, che devono confrontarsi con un panorama educativo complesso e in continua evoluzione.

Il Problema dell'Orientamento Universitario

L'orientamento universitario comporta diverse sfide:

- **Scelta del percorso di studi:** valutare le proprie attitudini, interessi e performance scolastiche per identificare il corso di studi più adatto.
- **Informazione e supporto:** la mancanza di un riferimento strutturato può rendere difficoltosa la scelta, lasciando gli studenti senza un adeguato sostegno informativo.
- **Adattamento al mercato del lavoro:** in un contesto in rapida evoluzione, la decisione sul percorso formativo deve tener conto anche delle future opportunità professionali.

L'utilizzo dei modelli generativi offre l'opportunità di supportare il processo decisionale, fornendo analisi approfondite basate sui dati individuali e sulle tendenze del mercato, contribuendo a orientamenti più informati e personalizzati.

Modelli Generativi e la loro Configurazione

I modelli generativi utilizzano algoritmi di intelligenza artificiale per creare contenuti testuali partendo da un input specifico. La configurazione del modello si basa su parametri chiave:

- **Temperature:** controlla il grado di casualità e creatività dell'output. Valori alti generano risultati più variabili.

- **Top_p e Top_k:** limitano il ventaglio delle possibili scelte durante la generazione, indirizzando il modello verso risposte più coerenti.
- **Max_output_tokens:** definisce il numero massimo di token, ovvero le unità base del testo, che il modello può produrre.

La gestione delle chiavi API avviene tramite variabile d'ambiente, per garantire l'accesso protetto ai servizi di intelligenza artificiale.

Esempio di Codice

Di seguito viene riportato un esempio in Python che illustra come configurare e utilizzare un modello generativo per fornire suggerimenti formali sull'orientamento universitario:

Listato A.1: Esempio di utilizzo di un modello generativo per l'orientamento universitario

```
import os
import google.generativeai as genai

# Configurazione dei parametri del modello generativo
config = genai.types.GenerationConfig(
    temperature=1.5,  # Regola la creativita del testo
    top_p=0.7,        # Limita il ventaglio delle scelte possibili
    top_k=1,          # Seleziona la scelta piu probabile
    max_output_tokens=100  # Limite massimo per la lunghezza del testo generato
)

def genera_suggerimenti(voti):
    # Configurazione del client con la chiave API sicura
    genai.configure(api_key=os.environ.get("GEMINI_API_KEY"))
    modello = genai.GenerativeModel("gemini-1.5-flash", generation_config=config)

    # Definizione del prompt per la richiesta di orientamento
    prompt = f"Fornisci un consiglio formale per l'orientamento universitario. Voti: {voti}"
    risposta = modello.generate_content(prompt)
    return risposta.text
```