ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea in Informatica

Analisi e visualizzazione della compatibilità tra emendamenti della Camera dei Deputati e sentenze della Corte costituzionale

Relatore:
Chiar.mo Prof.
Fabio Vitali

Correlatore:
Chiar.ma Prof.ssa
Monica Palmirani

Presentata da:
Manuel Flagelli

III Sessione

Anno Accademico 2023/2024

LegalXML,

Embedding semantico,

Similarità del coseno

Abstract

La presente tesi propone un'analisi della compatibilità semantica tra gli emendamenti discussi presso la Camera dei Deputati e le decisioni della Corte costituzionale, utilizzando tecniche avanzate di intelligenza artificiale e lo standard Akoma Ntoso per la rappresentazione strutturata dei documenti legali. Il lavoro si articola in diverse fasi: la raccolta e la conversione degli emendamenti tramite *scraping* e trasformazioni XSLT; l'estrazione delle sezioni rilevanti delle sentenze, identificate in collaborazione con un esperto giuridico; e la creazione di rappresentazioni vettoriali dense dei testi attraverso il modello di intelligenza artificiale bge-m3. Queste rappresentazioni consentono il calcolo del coseno di similarità, una metrica in grado di misurare la vicinanza semantica tra i documenti. L'analisi ha evidenziato una distribuzione moderata dei valori di similarità, con una media di 0.5395 e una deviazione standard di 0.0483. Sono stati individuati casi estremi di alta e bassa similarità, dai quali sono emerse riflessioni sulla coerenza tematica e sulle direzioni di pensiero espresse nei documenti. Il sistema sviluppato offre uno strumento innovativo per supportare l'attività legislativa e giudiziaria, facilitando l'identificazione di emendamenti potenzialmente incostituzionali o in conflitto con precedenti normativi. Sebbene i risultati ottenuti dimostrino l'efficacia dell'approccio adottato, sono presenti margini di miglioramento, come l'integrazione di tecniche avanzate di *multi-layered embedding*, *aboutness* e *semantic chunking*, per una rappresentazione più granulare e precisa dei documenti. Questo lavoro rappresenta un importante contributo verso l'uso dell'intelligenza artificiale per migliorare l'efficienza e la qualità delle attività legislative, aprendo la strada a ulteriori sviluppi nell'analisi automatizzata dei testi giuridici.

Indice

Capitolo 1

INTRODUZIONE

Negli ultimi anni, la tecnologia ha trasformato profondamente molti ambiti, introducendo strumenti in grado di migliorare il modo in cui affrontiamo le attività complesse. L'intelligenza artificiale (IA), in particolare, ha dimostrato di poter offrire grandi benefici anche in ambiti complicati come il diritto, un settore in cui ogni parola può avere diverse interpretazioni e ogni norma è legata a un contesto più ampio, migliorando l'efficienza dei processi e rendendo più semplice analizzare grandi volumi di informazioni. Nel contesto legale, l'IA trova applicazioni importanti come la classificazione di documenti, il supporto nella redazione legislativa e, come trattato nel presente lavoro, la ricerca di connessioni tra emendamenti e sentenze. Tuttavia, l'adozione di queste tecnologie non è priva di difficoltà: il diritto è complesso, le norme cambiano nel tempo e viene richiesta una comprensione profonda dei concetti giuridici. Proprio per questo, l'uso dell'IA in ambito giuridico richiede approcci specifici, in grado di comprendere non solo il contenuto semantico dei testi, ma anche di considerare gli aspetti specifici dell'ambito legale. Questa tesi si concentra sull'analisi della compatibilità tra gli emendamenti proposti presso la Camera dei Deputati e le decisioni della Corte costituzionale; l'obiettivo principale è capire se gli emendamenti seguono i principi già sanciti dalla giurisprudenza costituzionale, o se invece si allontanano, rischiando potenzialmente di risultare incostituzionali. Si tratta di un'area ancora poco studiata, ma con un forte potenziale per migliorare la qualità e la coerenza delle leggi. Il lavoro si basa sull'uso dello standard Akoma Ntoso, un formato pensato per rappresentare i documenti legali in

modo strutturato e in modo che possano essere analizzati facilmente dai modelli di IA. Per rappresentare i testi in formato numerico, ho utilizzato il modello bge-m3, che trasforma documenti in vettori densi, ovvero insiemi di numeri che catturano il loro significato semantico; confrontando questi vettori, è possibile calcolare quanto un emendamento sia simile a una sentenza, non solo a livello testuale, ma anche nel contenuto. Il progetto si sviluppa in diverse fasi: dalla raccolta e preparazione dei dati, alla loro trasformazione in un formato interpretabile dai modelli di IA, fino al calcolo delle similarità tra i documenti. I risultati ottenuti permettono di individuare somiglianze e differenze tra emendamenti e decisioni della Corte, offrendo uno strumento per valutare l'allineamento delle proposte legislative con la giurisprudenza esistente e identificando eventuali casi di incostituzionalità. Questo lavoro dimostra come l'uso dell'intelligenza artificiale possa supportare il processo legislativo, migliorandone l'efficienza e facilitando l'analisi delle proposte di legge in relazione alla giurisprudenza costituzionale. Pur essendoci ancora margini di miglioramento, il sistema descritto rappresenta un passo importante verso l'integrazione dell' intelligenza artificiale nel diritto, ponendo le basi per applicazioni future che possano migliorare il processo legislativo.

Capitolo 2

Stato dell'arte e IA nel settore legale

2.1 Stato dell'arte

Negli ultimi anni, l'intelligenza artificiale (IA) ha ricoperto un ruolo sempre più importante in vari settori, migliorando i processi tradizionali: il settore legale non fa eccezione a questa rivoluzione tecnologica: nel contesto legale l'IA offre strumenti che permettono di analizzare grandi quantità di dati, rendere autonomi processi ripetitivi ed estrarre informazioni rilevanti da testi complessi. Tra le principali applicazioni dell'IA nel settore legale, è presente la categorizzazione semantica delle decisioni giudiziarie: questo approccio, reso possibile grazie a tecniche di *machine learning* e di elaborazione del linguaggio naturale (Natural Language Processing - NLP), consente di classificare automaticamente le decisioni in base all'argomento trattato, il ragionamento giuridico, le citazioni normative e la rilevanza [1]. In questo modo, diventa più facile per i professionisti del diritto individuare le decisioni pertinenti e confrontare casi simili tra di loro, migliorando la qualità e la facilità d'accesso alle informazioni giuridiche. Un altro esempio è l'uso dell'IA nel processo di redazione legislativa, come avviene nel sistema LEOS (Legislation Editing Open Software) [3] sviluppato dalla commissione europea. LEOS integra strumenti basati sull'intelligenza artificiale per supportare i redattori nella creazione e nella revisione dei testi, suggerendo riferimenti normativi basandosi sulla similarità semantica con

il contenuto con cui si sta lavorando e tenendo anche conto della validità temporale delle disposizioni. In questo contesto l'IA non si limita a generare nuovi contenuti, ma guida il redattore attraverso suggerimenti, permettendo di ridurre gli errori e velocizzare il processo. Un'altra applicazione riguarda l'implementazione delle direttive europee nelle legislazioni nazionali, che, tramite un approccio ibrido dell'IA che combina tecniche simboliche e subsimboliche, rende possibile analizzare le somiglianze tra le direttive europee e le corrispondenti normative nazionali [2]. In questo conteso lo standard Akoma Ntoso, che approfondirò nella sezione 2.4, fornisce una struttura documentale che collega semanticamente i vari elementi di un testo giuridico. Grazie a queste tecnologie è possibile individuare eventuali discrepanze tra il testo della direttiva e la sua effettiva attuazione nelle nazioni.

2.2 Vantaggi, sfide e questioni di ricerca

Come discusso nella sezione precedente, l'integrazione dell'IA nel settore legale presenta importanti vantaggi come la riduzione del tempo, degli errori e delle risorse necessarie durante lo svolgimento di numerose attività legali. Tuttavia, sono presenti anche diverse sfide e questioni di ricerca a cui far fronte [2]:

- Il diritto non è solo un insieme di regole, ma comprende anche valori e principi difficili da ridurre a formule statiche.
- La rigidità delle norme codificate potrebbe impedire l'adattamento flessibile delle leggi all'evoluzione della società, rendendo necessario un equilibrio tra norme fisse e contesti variabili.
- Le lingue artificiali (come i linguaggi di programmazione) sono solo un sottoinsieme del linguaggio naturale; si deve quindi tener conto delle limitazioni che questo comporta nel contesto legale.
- Le norme potrebbero contenere intenzionalmente delle contraddizioni per bilanciare diversi interessi e istituzioni (pluralismo giuridico).
- Le previsioni basate sui dati passati non possono rilevare nuovi concetti giuridici emergenti o prevedere comportamenti futuri in modo adeguato, specialmente in contesti sociali in evoluzione.

2.3 IA ibrida

Una tecnica utilizzata per affrontare le sfide citate è l'approccio dell'intelligenza artificiale ibrida, la quale combina diverse tecniche per affrontare le sfide elencate in precedenza: i principi *"human-in-the-loop"*, *"human-on-the-loop"* e *"human-in-command"* vengono combinati usando tecniche di IA simbolica e subsimbolica. Essi rappresentano tre modi diversi di integrare l'intervento umano con l'IA, assicurando che quest'ultima venga usata in modo sicuro, trasparente ed etico ed evitando di cederle troppo potere decisionale.

Human-in-the-Loop

L'umano è attivamente coinvolto nel processo decisionale del sistema di IA; essa fornisce un risultato, ma l'operatore umano ha l'ultima parola. Questo primo approccio viene usato, per esempio, per decisioni che richiedono una revisione approfondita da parte di esperti legali.

Human-on-the-Loop

L'umano monitora il sistema di IA, ma non interviene direttamente in ogni decisione; il sistema di IA può prendere autonomamente delle decisioni, ma l'operatore umano ha la possibilità di intervenire se necessario. Questo secondo approccio viene usato, per esempio in contesti in cui l'IA può operare in autonomia, ma è comunque necessaria una supervisione umana per assicurarsi che non si verifichino errori.

Human-in-Command

L'umano ha pieno controllo del sistema di IA e può interrompere o modificare le azioni del sistema in qualsiasi momento (questo non implica un coinvolgimento continuo su ogni decisione, ma l'umano, quando vuole, può prendere il controllo). Questo terzo approccio è molto importante in situazioni come l'uso di sistemi di giustizia predittiva, dove l'IA può suggerire, per esempio, raccomandazioni sul rischio di recidiva di un imputato, ma il giudice può decidere di ignorare il suggerimento tenendo conto di fattori che l'IA non può valutare (come l'atteggiamento dell'imputato durante il processo).

L'IA simbolica si basa su regole e logiche formali e prende decisioni usando simboli e regole predefinite. È un tipo di IA spiegabile perché segue regole comprensibili e può giustificare ogni decisione che prende. L'IA subsimbolica, d'altra parte, non si basa su regole esplicite, ma su modelli statistici e algoritmi di apprendimento automatico (*machine learning*). Questo tipo di IA "impara" dai dati, riconosce pattern e fa previsioni basandosi su ciò che ha appreso, senza seguire regole predefinite. È molto utile per analizzare grandi quantità di dati, ma è poco trasparente e spiegabile. Le tecniche subsimboliche presentano importanti limiti quando applicate al diritto; questo approccio, pur essendo efficace nell'elaborare grandi quantità di dati, potrebbe risultare insufficiente per cogliere il contesto giuridico e il significato normativo. In particolare, i principali problemi sono i seguenti [2]:

- Granularità vs Struttura: il *machine learning* lavora a livello di segmenti di testo, ma non riesce a collegare in modo semantico parti differenti del discorso giuridico, come l'obbligo e l'eccezione o il dovere e la sanzione. È quindi necessario un livello simbolico, basato su regole, per collegare questi elementi.
- Contenuto vs Contesto: il *machine learning* spesso analizza il contenuto senza tenere conto del contesto, portando a interpretazioni errate. Ad esempio, il lessico legale cambia nel tempo e senza considerare il contesto giuridico e temporale, si possono fare deduzioni errate.
- Passato vs Futuro: gli algoritmi di *machine learning* si basano su dati storici; quindi, non possono interpretare correttamente nuovi concetti introdotti da nuove leggi. Serve un'analisi semantica per gestire questi nuovi concetti.
- Statico vs Dinamico: le leggi cambiano nel tempo e l'IA deve tenere conto delle modifiche normative, delle abrogazioni e delle versioni successive di un testo giuridico.

2.4 Il ruolo di Akoma Ntoso

Lo standard XML (eXtensible Markup Language) Akoma Ntoso (AKN), in quanto componente simbolica, rappresenta una soluzione fondamentale per risolvere molte delle sfide che emergono dall'adozione dell'IA nel settore legale. Grazie alla sua struttura formale, Akoma Ntoso permette di rappresentare metadati e contenuti dei

testi legali, consentendo di collegare semantica e contenuti giuridici e di organizzare il testo in sezioni che collegano in modo esplicito i concetti chiave, offrendo una maggiore granularità e chiarezza logica. Nel contesto dell'IA Ibrida, che combina tecniche simboliche e subsimboliche per affrontare le complessità del diritto, Akoma Ntoso supporta la contestualizzazione dei riferimenti legali, integrando contesto giuridico e temporale per una migliore comprensione delle giurisdizioni e delle modifiche normative nel tempo. In questo modo, i sistemi di IA possono non solo estrarre dati dai testi giuridici, ma anche comprenderne il contesto e l'evoluzione normativa, superando i limiti dell'IA subsimbolica.

Nonostante gli importanti progressi che ho descritto nell'ambito dell'applicazione dell'IA nel settore legale, nessuno studio si è concentrato sull'analisi della compatibilità tra emendamenti della Camera dei Deputati e decisioni della Corte costituzionale. Il presente lavoro ha l'obiettivo di colmare questa lacuna sviluppando un sistema che utilizza l'IA ibrida per analizzare la similarità semantica tra emendamenti e sentenze, sfruttando le potenzialità di Akoma Ntoso per rappresentare i documenti. Lo scopo è quello di comprendere meglio se le nuove proposte legislative convergano o divergano rispetto alla giurisprudenza esistente, portando una migliore comprensione giuridica e maggiore efficienza nel processo legislativo.

Capitolo 3

Preparazione del Dataset

In questo capitolo viene descritta la preparazione del dataset, costituito da un insieme di emendamenti della Camera dei Deputati e sentenze delle Corte costituzionale. In particolare, gli emendamenti con cui ho lavorato, sono quelli relativi a 6 Proposte di Legge discusse presso la Camera dei deputati: AC1321, AC1632, AC1660, AC1896, AC1902 e AC1937; mentre le sentenze prese in considerazione sono quelle pronunciate dal 14/06/1956 al 10/05/2024.

3.1 Emendamenti della Camera dei Deputati

I passaggi per costruire il dataset degli emendamenti sono il processo di *scraping* degli emendamenti dal sito della Camera dei deputati e la loro successiva conversione nel formato AKN (Akoma Ntoso) tramite l'uso di uno *stylesheet* XSLT (*eXtensible Stylesheet Language Transformations*). Queste fasi permettono di ottenere un dataset strutturato e adatto all'analisi semantica.

3.1.1 Scraping degli Emendamenti

La prima fase consiste nell'estrazione automatizzata dei dati dal sito della Camera dei deputati per scaricare gli emendamenti in formato HTML (HyperText Markup Language) tramite uno script nel linguaggio di programmazione Python.

Il processo di *scraping* inizia dalla costruzione di *Uniform Resource Locator* (URL) personalizzati (seguendo lo schema degli URL del sito della Camera) per ogni lista di emendamenti di ciascuna Proposta di Legge e adattando i link alla specifica legislatura e al numero di atto legislativo. Per questo scopo ho usato dei *template* di URL ai quali è stato possibile sostituire dinamicamente i valori della legislatura e del numero dell'atto. Ad esempio, l'URL della lista di emendamenti relativi a un atto e a una legislatura specifici: "`https://documenti.camera.it/apps/emendamenti/getProposteEmendative.aspx?contenitorePortante=leg.{legislatura}.eme.ac.{n_atto}&tipoSeduta=1&sedeEsame=referente&urnTestoRiferimento=urn:leg:{legislatura}:{n_atto}:null:null:com:66:referente&tipoListaEmendamenti=1`"

In seguito, ho creato un file JSON (JavaScript Object Notation) contenente un dizionario con tutti i link (seguendo il *template* indicato sopra) delle liste di emendamenti per ciascuna delle proposta di legge candidate, attraverso i quali il codice può accedere alle pagine dei documenti (tramite richieste HTTP - HyperText Transfer Protocol) e recuperarne il contenuto. Ho utilizzato la libreria "BeautifulSoup" [8] per effettuare il parsing dell'HTML, navigare tra i tag e, dopo aver analizzato e studiato attentamente la struttura HTML delle pagine web degli emendamenti, selezionare quelli pertinenti, ovvero i tag in cui erano contenute le informazioni rilevanti degli emendamenti. Ho, quindi, esportato i dati relativi a ciascuna proposta emendatva, ciascuno in un file il cui nome corrisponde al numero dell'emendamento (es: 1_1, 1_2...), contenuti all'interno di cartelle denominate con il numero della Proposta di Legge.

Di seguito allego il codice relativo allo *scraping* degli emendamenti appartenenti alla 18° Legislatura (AC1321); il funzionamento per gli emendamenti delle Proposte di Legge appartenenti alla 19° Legislatura è analogo, con le uniche differenze dei nomi di alcuni tag, classi e id nell'HTML:

```
1 def scrape_edc18(atti, resume=None):
2     #riprendi da un punto specifico, se indicato
3     if resume:
4         resume_idx = list(atti.keys()).index(str(resume))
```

```
5           remaining_atti =
            ↪ dict(list(OrderedDict(atti.items()).items())[resume_idx:])
6
7       #itera sugli tutti atti (o quelli rimanenti se resume è fornito)
8       for n_atto in tqdm(remaining_atti if resume else atti, desc="scraping
        ↪ atto", initial=resume_idx if resume else 0, total=len(atti)):
9           path_atto = Path(f"EDC_HTMLdata/Leg18/Atto{n_atto}")
10          if path_atto.exists():
11              continue
12
13          #richiede la pagina web dell'atto
14          edc_response = requests.get(atti[n_atto])
15          if edc_response.status_code == 200:
16              #analizza il contenuto della pagina
17              soup = BeautifulSoup(edc_response.content, 'lxml')
18              table = soup.find('tbody')
19              rows = table.find_all('tr')
20
21              #itera sulle righe della tabella della lista di emendamenti
22              for row in tqdm(rows, desc=f"scraping row of atto {n_atto}",
                ↪ leave=True):
23                  #crea una cartella per ogni articolo
24                  if "rigaArticoloNormale" in row['class']:
25                      art = row.find('a').text.replace(". ", "")
26                      path = path_atto / f"{art}"
27                      path.mkdir(parents=True, exist_ok=True)
28                  elif "normale" in row['class']:
29                      row_item = row.find('a', {'rel': "popup"})
30                      row_num = row_item.text[:-1].replace(".", "_")
31                      row_url =
                        ↪ "https://documenti.camera.it/apps/emendamenti/" +
                        ↪ str(row_item.attrs['href']).strip()
32                      row_path = path / f"{row_num}.html"
33
34                      #richiede e salva il contenuto della pagina del
                        ↪ singolo emendamento
```

```
35                          row_response = requests.get(row_url)
36                          if row_response.status_code == 200:
37                              row_soup = BeautifulSoup(row_response.content,
                                ↪  'lxml')
38                              sedute = row_soup.find(id="sedute")
39                              with open(str(row_path), 'w+', encoding='utf-8')
                                ↪  as file:
40                                  file.write(str(sedute))
```

3.1.2 Conversione degli HTML in AKN tramite XSLT

Una volta raccolti i dati in formato HTML, ho effettuato il processo di conversione utilizzando uno stylehseet XSLT nel formato AKN, progettato nello specifico per rappresentare documenti legali e rendenderli meglio interpretabili dal punto di vista semantico da modelli di *machine learning*.

XSLT è un linguaggio di trasformazione: dato un documento XML (o HTML), è possibile generare un altro documento XML derivato, applicando una serie di regole di trasformazione definite nello *stylesheet*. Sfrutta XPath (XML Path Language) [14], un linguaggio di selezione, per navigare e selezionare elementi all'interno del documento XML. XSLT è molto efficace nel ristrutturare un documento e creare nuovi contenuti: nel progetto ho usato XSLT per convertire i file HTML, che contenevano soltanto il corpo dell'emendamento, in un file XML che rispetta la sintassi di Akoma Ntoso (elementi come *"meta"*, *"preface"*, *"amendmentBody"* e altri). Per eseguire la conversione da HTML a AKN, ho scelto di utilizzare "Saxon" [9], un processore XSLT molto avanzato, compatibile con XSLT 3.0. Ho compiuto questa scelta per le sue ottime prestazioni nel processare grandi volumi di dati XML (come nel mio caso); per la documentazione completa e chiara; e infine, come approfondito alla fine di questo capitolo, per possibilità di sfruttare alcune funzionalità avanzate dello *stylesheet* XSLT usato per la conversione, che non sarebbero state supportate da altri strumenti meno avanzati (come "lxml")[13].

Come accennato in precedenza, un elemento fondamentale per la trasformazione è il foglio di stile XSLT, che consiste in un documento XML che contiene regole di trasformazione espresse attraverso dei *"template"*. Ogni *"template"* descrive come

deve essere processato un nodo specifico dell'HTML di input (come un elemento o un attributo). Il foglio di stile contiene diversi elementi chiave, tra cui: gli elementi *"xsl:template"* e *"xsl:apply-templates"*, che permettono di definire le regole di trasformazione; elementi condizionali come *"xsl:if"* e *"xsl:choose"*, che consentono di creare trasformazioni dinamiche in base alla logica condizionale; l'attributo *"xsl:match"*, compatibile con XSLT 2.0 o superiore, permette di selezionare dei nodi di specifici; infine, elementi come *"xsl:element"* e *"xsl:value-of"* consentono, rispettivamente, di creare nuovi elementi XML ed estrarre il contenuto di nodi selezionati.

Di seguito allego alcuni estratti dello *stylesheet* che mostrano come ho utilizzato gli elementi XSLT citati in precedenza:

```
<!-- Controlla se "Il Relatore" è presente tra i proponenti-->
<xsl:if test="contains(//div[@class='sedutaNuova']//div[@class='propostaE
↪ mendativa']//div[@class='proponenti'], 'Il Relatore')">
    <TLCPerson id="relatore" href="/ontology/role/relatore" showAs="Il
    ↪ Relatore"/>
</xsl:if>
```

```
<xsl:variable name="dataPresentazione">
    <xsl:choose>
        <!-- Cerca il primo div[@class='testata'] all'interno di
        ↪ div[@class='sedutaNuova'] -->
        <xsl:when test="count(//div[@class='sedutaNuova']//div[@class='te
        ↪ stata']) > 0">
            <xsl:value-of select="substring-before(substring-after(//div[
            ↪ @class='sedutaNuova']//div[@class='testata'][1], 'del '),
            ↪ ' ')"/>
        </xsl:when>
        <!-- Cerca un div[@class='testata'] generico (fallback) -->
        <xsl:otherwise>
            <xsl:value-of select="substring-before(substring-after(//div[
            ↪ @class='testata'][1], 'del '), ' ')"/>
        </xsl:otherwise>
    </xsl:choose>
```

```
</xsl:variable>
```

Infine, ho implementato la funzione *"convert_html_to_akn"*, che automatizza la conversione dei file HTML in formato XML (standard Akoma Ntoso). Prende in input il percorso del file HTML, lo *stylesheet* XSLT, i parametri "numero_atto" e "numero_legislatura" (che vengono ricavati tramite una funzione esterna), il percorso del file .jar di Saxon e una cartella di output. La funzione costruisce il comando ed esegue Saxon, generando il file XML.

```
def convert_html_to_akn(html_file, xslt_file, numero_atto,
↪  numero_legislatura, saxon_jar_path, output_folder):
    relative_path = os.path.relpath(html_file, start=base_folder)
    output_file = os.path.join(output_folder,
    ↪  relative_path).replace(".html", ".akn.xml")

    #crea la cartella di output
    os.makedirs(os.path.dirname(output_file), exist_ok=True)

    #costruisce il comando da eseguire
    cmd = [
        'java', '-jar', saxon_jar_path,
        f'-s:{html_file}',
        f'-xsl:{xslt_file}',
        f'-o:{output_file}',
        f'numero_atto={numero_atto}',
        f'numero_legislatura={numero_legislatura}'
    ]

    #esegue il comando
    result = subprocess.run(cmd, stdout=subprocess.PIPE,
    ↪  stderr=subprocess.PIPE, text=True)

    #gestisce i casi di successo ed errore
    if result.returncode == 0:
```

```
23          print(f"Converted {html_file} to {output_file} with
            ↪  numero_atto={numero_atto} and
            ↪  numero_legislatura={numero_legislatura}")
24      else:
25          print(f"Error converting {html_file}: {result.stderr}")
```

3.2 Sentenze della Corte costituzionale

Le sentenze della corte costituzionale mi sono state fornite in formato Akoma Ntoso dalla prof.ssa Monica Palmirani. È stato, però, necessario un processo di estrazione delle informazioni rilevanti per il confronto con gli emendamenti, al fine ridurre il rumore che la grande quantità di informazioni presente in ciascuna sentenza avrebbe generato nel calcolo della similarità.

3.2.1 Individuazione delle sezioni rilevanti

Le sezioni delle sentenze, rilevanti per il confronto con gli emendamenti, individuate dal costituzionalista Pier Francesco Bresciani sono:

- L'**epigrafe**, che contiene: l'identificazione del tipo di provvedimento giurisdizionale (sentenza o ordinanza); il tipo di giudizio in cui la sentenza è pronunciata (legittimità costituzionale, conflitto di attribuzione o ammissibilità di referendum abrogativo, in questo contesto sono rilevanti solo le sentenze pronunciate in giudizio di legittimità costituzionale); l'oggetto del giudizio (la norma scrutinata); la modalità di instaurazione del giudizio (in via incidentale, ovvero sollevato da un giudice in processo, o in via principale, ovvero proposto da una Regione contro lo Stato o viceversa). Ho individuato questi temi nei 2 elementi XML <tipo_procedimento> e <introduction>.
- I primi due punti del **Considerato in Diritto**, che ho individuato nell'elemento XML <motivation>, i quali contengono in maniera sintetica il contenuto della norma oggetto di scrutinio della Corte e i motivi per cui il giudice a quo la ritiene potenzialmente in contrasto con la Costituzione.
- Il **Dispositivo**, individuato nell'elemento XML <decision>, che contiene delle formule che permettono di capire se esiste un contrasto ("Dichiara l'illegitti-

mità costituzionale...”), non esiste un contrasto (“Dichiara non fondate...”) o altri casi.

3.2.2 Estrazione delle sezioni rilevanti

Per estrarre dalle sentenze le sezioni rilevanti per il confronto con gli emendamenti, ho scritto un codice in Python per automatizzare il processo. Ho usato la libreria “lxml” [13] per parsare i documenti delle sentenze ed effettuare un controllo, tramite query XPath, dell'elemento <tipo_procedimento> sul tipo di giudizio in cui la sentenza è stata pronunciata, scartando le sentenze pronunciate in giudizi diversi da quello di legittimità. Successivamente, ho individuato gli elementi descritti nella sezione precedente e ho estratto il contenuto degli elementi <introduction>, <tipo_procedimento>, <decision> e <motivation>; per quest'ultimo elemento, contenente il “Considerato in Diritto” ho implementato un algoritmo per selezionare solo i primi due punti rilevanti.

Nel seguente estratto di codice è presente l'implementazione del sistema descritta nel paragrafo precedente:

```
#estrazione dell'epigrafe
epigrafe = root.find('.//akn:judgmentBody/akn:introduction', namespaces)
if epigrafe is not None:
    epigrafe_content = etree.tostring(epigrafe, encoding='unicode', method='xml')
    epigrafe_content = remove_namespaces(epigrafe_content)
    extracted_data["epigrafe"] = f"<tipo_procedimento>{tipo_procedimento_content}</tipo_procedimento>\n{epigrafe_content}"
print(f"Epigrafe trovata: {epigrafe is not None}")

#estrazione dell'elemento <motivation>
motivation = root.find('.//akn:judgmentBody/akn:motivation', namespaces)
if motivation is not None:
    print("Trovato nodo motivation.")
```

```
13      paragraphs = motivation.xpath('.//akn:paragraph',
        ↪  namespaces=namespaces)
14      extracted_paragraphs = []
15      capture = False
16      captured_nums = set()
17
18      #algoritmo che seleziona solo i primi due punti
19      for paragraph in paragraphs:
20          num = paragraph.find('.//akn:num', namespaces=namespaces)
21
22          #se viene individuato il punto 1 o 2, avvia la modalità di
            ↪  cattura
23          if num is not None and num.text.strip() in {"1.", "2."}:
24              if num.text.strip() not in captured_nums:
25                  capture = True
26                  captured_nums.add(num.text.strip())
27                  extracted_paragraphs.append(etree.tostring(paragraph,
                    ↪  encoding='unicode', method='xml'))
28          elif capture:
29              next_num = paragraph.find('.//akn:num',
                ↪  namespaces=namespaces)
30              #se il prossimo numero è diverso da 1 o 2, disattiva la
                ↪  modalità di cattura
31              if next_num is not None and next_num.text.strip() not in
                ↪  {"1.", "2."}:
32                  capture = False
33              if capture:
34                  extracted_paragraphs.append(etree.tostring(paragraph,
                    ↪  encoding='unicode', method='xml'))
35
36          #se sono stati estratti il punto 1 e 2 ed è stato individuato
            ↪  che il prossimo paragrafo contiene un nuovo punto, ferma
            ↪  il ciclo
37          if len(captured_nums) == 2 and not capture:
38              break
39
```

```
    #combina i paragrafi estratti (tutti appartenenti ai numeri 1 e 2)
    if extracted_paragraphs:
        combined_content = ''.join(remove_namespaces(p) for p in extracted_paragraphs)
        extracted_data["considerato_in_diritto"] = combined_content
    else:
        print("Meno di due punti trovati in motivation.")
else:
    print("Nodo motivation non trovato.")

#estrazione del dispositivo
dispositivo = root.find('.//akn:judgmentBody/akn:decision', namespaces)
if dispositivo is not None:
    dispositivo_content = etree.tostring(dispositivo, encoding='unicode', method='xml')
    dispositivo_content = remove_namespaces(dispositivo_content)
    extracted_data["dispositivo"] = dispositivo_content
print(f"Dispositivo trovato: {dispositivo is not None}")
```

Una volta completata la conversione, verificato la conformità della struttura degli emendamenti agli standard di Akoma Ntoso e aver estratto dalle sentenze le sezioni rilevanti per il confronto con gli emendamenti, il dataset è completo e pronto per essere elaborato dal modello di intelligenza artificiale.

Capitolo 4

Modello IA e creazione dei vettori

4.1 Vantaggi di Bge-m3 e FlagEmbedding

In seguito alla preparazione e conversione degli emendamenti in formato Akoma Ntoso e all'estrazione delle sezioni rilevanti delle sentenze per il confronto con gli emendamenti, è stato possibile procedere alla fase di *embedding*. In questa fase ho sfruttato un modello di intelligenza artificiale per rappresentare i testi degli emendamenti della Camera e le decisioni della Corte sotto forma di vettori densi, che supportano l'applicazione di una metrica (coseno di similarità) per misurare la similarità semantica. A questo scopo, ho scelto di sfruttare le potenzialità di "bge-m3" [7], un modello di *embedding* testuale sviluppato dal Beijing Academy of Artificial Intelligence (BAAI). È caratterizzato da un supporto multilingua, è pre-addestrato e si basa su tecniche di *machine learning* avanzate riuscendo così a catturare in modo efficace il contenuto semantico dei testi. Bge-m3 è progettato per supportare multi-granularità, ovvero la capacità di gestire le informazioni sia a livello di singole parole, sia a livello di frasi o documenti, il che lo rende particolarmente adatto per rappresentazioni semantiche complesse. Un altro dei punti di forza di bge-m3 consiste nel poter comprendere non solo il significato letterale delle parole, ma anche le relazioni tra concetti e contesti: questa capacità deriva dall'uso di self knowledge distillation [6], una tecnica che, in fase di training, prevede che un modello "apprenda da sé", utilizzando il proprio output come "ground truth" per affinare le rappresentazioni correggendo errori e migliorando la capacità di catturare relazioni

più profonde tra i dati. Questa caratteristica è fondamentale nel contesto giuridico, dove sia le parole che le frasi possono avere significati diversi in base al contesto. Inoltre, grazie al suo design, bge-m3 è in grado di creare rappresentazioni vettoriali ad alta dimensionalità (codificando molte caratteristiche come il contesto, il tono, le relazioni tra parole...) in tempi relativamente rapidi [6], che è un aspetto molto importante in questo contesto, considerata la vastità dei dataset. Infine, in quanto modello pre-addestrato, mi ha permesso di reindirizzare le risorse che avrei dovuto impiegare nell'addestramento del modello, verso altri aspetti importanti del progetto, come l'ottimizzazione del dataset. Per applicare il modello bge-m3 e gestire i vettori, ho utilizzato la libreria FlagEmbedding [11], una libreria progettata per supportare l'uso di modelli NLP (Natural Language Processing) e vettorializzazioni complesse. Una caratteristica importante di questa libreria è la capacità di supportare, grazie all'integrazione con bge-m3, la creazione di vettori fino a 8192 token. I token sono le unità fondamentali in cui il testo viene suddiviso per essere processato da un modello di intelligenza artificiale; in base al modello, un token può corrispondere a una parola, una parte di parola e in alcuni casi anche a un carattere. La possibilità di rappresentare dei documenti con un massimo di 8192 token consente di catturare una grande vastità di dettagli e informazioni rilevanti, senza la necessità di implementare tecniche di segmentazione (concetto approfondito nel capitolo 6).

4.2 Preprocessing dei testi

Un passaggio molto importante che ho dovuto compiere prima del processo di *embedding*, è stata la preparazione dei testi degli emendamenti affinché fossero contestualizzati relativamente all'articolo e ai commi originali che intendono emendare; è un passaggio fondamentale per rendere completa la comprensione dell'emendamento da parte del modello. Questo passaggio si è reso necessario perché i miei files Akoma Ntoso, per semplificare il processo di conversione, non contengono i riferimenti, i quali indirizzerebbero automaticamente il modello verso la sezione normativa che l'emendamento vuole modificare. Nel mio caso, non avendo a disposizione i riferimenti, ho implementato un sistema che crea, per ogni emendamento, un "contenuto combinato", ovvero un testo che contiene l'emendamento e l'articolo che intende modificare, fornendo al modello un contesto completo. Il primo passaggio è stato quello

di estrarre dall'emendamento in formato Akoma Ntoso, il numero del Progetto di Legge e dell'articolo che viene emendato.

Il seguente codice mostra l'estrazione dal testo dell'emendamento delle informazioni necessarie per reperire l'articolo relativo. "AffectedDocument" rappresenta il numero del Progetto di Legge e "docNumber" il numero identificativo dell'emendamento, la cui parte intera indica il numero dell'articolo emendato.

```
def extract_doc_info(file_path):
    tree = etree.parse(file_path)
    namespaces = {'akn': 'http://docs.oasis-open.org/legaldocml/ns/akn/3.0/CSD03'}

    # estrazione contenuto elemento <docNumber>. Esempio: "Emendamento n. 3.4"
    doc_number_full = tree.xpath('//akn:docNumber/text()', namespaces=namespaces)
    full_doc_number = doc_number_full[0].strip() if doc_number_full else "Numero non trovato"

    # estrazione numero articolo emendato. Esempio: "3"
    doc_number = re.search(r'\d+', full_doc_number).group() if full_doc_number != "Numero non trovato" else "Numero non trovato"

    # estrazione numero emendamento. Esempio: "3.4"
    doc_decimal_number = re.search(r'\d+(\.\d+)+', full_doc_number)
    doc_decimal_number = doc_decimal_number.group() if doc_decimal_number else "Numero decimale non trovato"

    # estrazione numero progetto di legge. Esempio: "AC1321"
    affected_element = tree.xpath('//akn:affectedDocument', namespaces=namespaces)
    affected_document = affected_element[0].text.strip() if affected_element and affected_element[0].text else "Documento non trovato"

```

```
20      return affected_document, doc_number, doc_decimal_number
```

In seguito ad aver analizzato la struttura dei Progetti di Legge (in formato HTML) e compreso i pattern delle classi e degli id che identificano i div (contenitori) che contengono gli articoli, ho estratto e unito questi ultimi con gli emendamenti in un unico file. Di seguito, allego un esempio di "contenuto combinato" dell'emendamento 11.2 relativo all'articolo 11 dell'Atto della Camera 1660:

```
1
2  AC1660 Art. 11.
3  (Modifiche all'articolo 1-bis del decreto legislativo 22 gennaio 1948, n.
   ↪  66, relativo all'impedimento della libera circolazione su strada)
4  1. All'articolo 1-bis, comma 1, del decreto legislativo 22 gennaio 1948,
   ↪  n. 66, sono apportate le seguenti modificazioni:
5    a) al primo periodo, dopo la parola: ≪ordinaria≫ sono inserite le
     ↪  seguenti: ≪o ferrata≫ e le parole: ≪con la sanzione amministrativa
     ↪  del pagamento di una somma da euro 1.000 a euro 4.000≫ sono
     ↪  sostituite dalle seguenti: ≪con la reclusione fino a un mese o la
     ↪  multa fino a 300 euro≫;
6    b) il secondo periodo è sostituito dal seguente: ≪La pena è della
     ↪  reclusione da sei mesi a due anni se il fatto è commesso da più
     ↪  persone riunite≫.
7
8  <akomaNtoso xmlns="http://docs.oasis-open.org/legaldocml/ns/akn/3.0"
9      xmlns:akn4it="http://akt4it.org/metadata"
10     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
11     xsi:schemaLocation="http://docs.oasis-open.org/legaldocml/ns/akn/3.0
       ↪  ./akomantoso30.xsd">
12    <amendment contains="originalVersion" name="emendamento">
13       <meta>
14          <identification source="#cirsfid-alma-ai">
15             <FRBRWork>
16                <FRBRthis value="/akn/it/camera/emendamento/2024-05-28/19-⌋
                  ↪  AC1660-11-2/!main"/>
17                <FRBRuri value="/akn/it/camera/emendamento/2024-05-28/19-A⌋
                  ↪  C1660-11-2"/>
```

```
            <FRBRalias name="" value=""/>
            <FRBRdate date="2024-05-28" name="presentazione"/>
            <FRBRauthor href="#camera" as="#author"/>
            <FRBRcountry value="it"/>
         </FRBRWork>
         <FRBRExpression>
            <FRBRthis value="/akn/it/camera/emendamento/2024-05-28/19-AC1660-11-2/@ita/!main"/>
            <FRBRuri value="/akn/it/camera/emendamento/2024-05-28/19-AC1660-11-2/@ita"/>
            <FRBRdate date="2024-05-28" name="presentazione"/>
            <FRBRauthor href="#camera" as="#author"/>
            <FRBRlanguage language="ita"/>
         </FRBRExpression>
         <FRBRManifestation>
            <FRBRthis value="/akn/it/camera/emendamento/2024-05-28/19-AC1660-11-2/@ita/!main.xml"/>
            <FRBRuri value="/akn/it/camera/emendamento/2024-05-28/19-AC1660-11-2/@ita/!main.akn"/>
            <FRBRdate date="2024-05-28" name="presentazione"/>
            <FRBRauthor href="#camera"/>
         </FRBRManifestation>
      </identification>
      <lifecycle source="#cirsfid-alma-ai">
         <eventRef source="#camera" eId="event1" date="2024-05-28"/>
      </lifecycle>
      <workflow source="#cirsfid-alma-ai">
         <step eId="step1" by="#person_1" date="2024-05-28" outcome="" as="memberOfParliament"/>
      </workflow>
      <analysis source="#cirsfid-alma-ai">
         <activeModifications/>
      </analysis>
      <references source="#cirsfid-alma-ai">
         <activeRef href="/akn/it/camera/atto/19/AC1660" showAs="AC AC1660"/>
```

```
48              <TLCOrganization eId="camera" showAs="Camera dei Deputati"
                ↪ href="/ontology/org/it/Camera"/>
49              <TLCRole eId="author" href="/ontology/role/author"
                ↪ showAs="Author"/>
50              <TLCPerson eId="person1" href="https://documenti.camera.it/ap⌋
                ↪ ps/commonServices/getDocumento.ashx?sezione=deputati&amp;⌋
                ↪ tipoDoc=schedaDeputato&amp;idLegislatura=19&amp;idPersona⌋
                ↪ =308929&amp;webType=Normale" showAs="Soumahoro
                ↪ Aboubakar"/>
51           </references>
52        </meta>
53        <preface>
54           <p>
55              <affectedDocument>AC1660</affectedDocument>
56           </p>
57           <p>
58              <docNumber>Emendamento n. 11.2</docNumber>
59           </p>
60           <p>
61              <docProponent refersTo="#person1" as="deputato">
62                 <span>Soumahoro Aboubakar</span>
63              </docProponent>
64           </p>
65        </preface>
66        <amendmentBody>
67           <amendmentContent>
68              <p>Sopprimerlo.</p>
69           </amendmentContent>
70        </amendmentBody>
71     </amendment>
72  </akomaNtoso>
```

In questo modo, i "contenuti combinati" sono strutturati in modo da fornire il contesto completo al modello di IA e sono quindi pronti per l'*embedding*.

4.3 Processo di Embedding

Per confrontare i documenti sulla base del loro contenuto semantico e non solo testuale, ho generato dei vettori densi a partire dai documenti; questo approccio consente di individuare eventuali somiglianze tra i testi, anche nel caso in cui parlano dello stesso tema, ma usando parole diverse. Un vettore denso è una rappresentazione numerica in uno spazio multidimensionale che viene usato per rappresentare il significato dei testi; ogni dimensione del vettore corrisponde a una caratteristica del documento come il contesto, il tono e le relazioni tra parole. Nel progetto, utilizzando il modello bge-m3 e la libreria FlagEmbedding, ho trasformato i testi combinati (emendamento e articolo relativo) e gli estratti delle sentenze in vettori densi, scegliendo con attenzione il parametro relativo al numero massimo di token da considerare per ciascun documento. In particolare, ho effettuato un'analisi del dataset dalla quale è emerso che tutti i documenti rientravano nella soglia limite di 8192 token: questo limite permette ai testi di essere rappresentati nella loro interezza e senza troncamenti, il che si traduce nella conservazione integrale del contenuto semantico.

Il seguente codice mostra la creazione del vettore di un testo combinato (emendamento e articolo relativo) e la conseguente estrazione dei dati identificativi dell'emendamento per salvarlo correttamente:

```
# generazione del vettore denso per il testo combinato
vector = model.encode([combined_text], batch_size=1,
↪  max_length=8192)['dense_vecs'][0]

# estrazione delle informazioni identificative dall'emendamento
affected_document, doc_number, full_doc_number =
↪  extract_doc_info(file_path)

# salvataggio del vettore
vector_filename = f"vector_{affected_document}_{full_doc_number}.npy"
vector_path = os.path.join(vectors_directory, vector_filename)
np.save(vector_path, vector)
```

Una volta generati i vettori sia per gli emendamenti che per le sentenze, è stato

possibile procedere con il calcolo della similarità tra i due tipi di documento.

Capitolo 5

Compatibilità semantica e coseno di similarità

5.1 Definizione

In questo capitolo verrà trattato il concetto di "similarità", un elemento molto importante per comprendere le correlazioni tra gli emendamenti proposti alla Camera dei Deputati e le decisioni della Corte costituzionale. L'obiettivo di questa analisi è evidenziare i punti di convergenza o divergenza tra le idee contenute nei testi degli emendamenti e quelle espresse nelle decisioni passate della Corte, per comprendere meglio come questi documenti possano riflettere o contrapporsi a orientamenti giuridici già consolidati. È importante sottolineare che il concetto di "similarità" non si riferisce semplicemente alla presenza di parole identiche o somiglianze superficiali, ma è un concetto più profondo, legato al significato semantico dei testi. A tal proposito, i documenti da confrontare vengono rappresentati in forma di vettori, che sono insiemi di numeri capaci di riflettere il contenuto semantico di un documento. Tra due vettori è possibile applicare una misura matematica chiamata "coseno di similarità". La similarità tra due vettori, nel caso del coseno di similarità, consiste nel calcolare l'angolo che li separa: un angolo più piccolo indica una maggiore somiglianza, mentre un angolo maggiore si traduce in una minore somiglianza. I valori del coseno di similarità variano tra -1 e 1, dove 1 indica similarità massima (i vettori hanno la stessa direzione), 0 segnala assenza totale di somiglianza (i vettori sono

perpendicolari) e -1 rappresenta massima dissimilarità (i vettori sono opposti).

5.2 Formula e funzionamento matematico

Il coseno di similarità tra due vettori A e B si calcola come segue:

$$\text{Coseno di similarità} = \frac{A \cdot B}{\|A\| \, \|B\|}$$

dove:

- $A \cdot B$ è il prodotto scalare dei due vettori;
- $\|A\| \, \|B\|$ è il prodotto delle norme euclidee (lunghezze) dei vettori A e B.

Una componente importante della formula è la normalizzazione, la quale permette di calcolare un valore di similarità coerente tra vettori di lunghezza diversa. Normalizzare un vettore significa ridurre la sua lunghezza a un valore unitario, non cambiando la sua direzione. Nel calcolo del coseno di similarità, la normalizzazione avviene dividendo ogni vettore per la sua norma euclidea (lunghezza), in modo che il vettore risultante abbia lunghezza 1. Senza di essa, i risultati sarebbero influenzati dalla lunghezza dei vettori: vettori di lunghezza diversa, ma con orientamento simile, potrebbero risultare meno simili di quanto lo sono realmente; vettori di lunghezza simile ma orientamento diverso potrebbero apparire simili anche senza esserlo. La normalizzazione è, quindi, fondamentale per la correttezza della similarità evitando che i valori vengano distorti. Nel nostro caso la formula si traduce in un valore che riflette quanto i testi degli emendamenti siano semanticamente vicini alle decisioni della Corte costituzionale, rappresentati anch'essi in forma di vettori. Una volta calcolato il coseno di similarità, è stato possibile analizzare i risultati per osservare le convergenze o le divergenze tra ogni emendamento e le decisioni.

5.3 Similarità coseno tra emendamenti e decisioni

Per il calcolo del coseno di similarità, ho utilizzato la funzione *"cosine_similarity"* di scikit-learn [10], una libreria di *machine learning* che offre funzioni pronte per l'uso per calcolare la similarità tra vettori. Una delle motivazioni che mi ha spinto a usare la funzione di scikit-learn al posto della semplice formula manuale è che

"cosine_similarity" è ottimizzata per eseguire calcoli vettoriali su larga scala, ottimizzando le prestazioni sui calcoli tra matrici: nel mio caso, con un dataset di grandi dimensioni e con la necessità di effettuare tantissimi confronti (più di 15 milioni), questo rappresenta un grande vantaggio. Inoltre, *"cosine_similarity"* normalizza automaticamente i vettori in input, eliminando i potenziali errori legati alla normalizzazione.

Di seguito allego il codice che ho implementato per calcolare la similarità tra le coppie di vettori di emendamenti e sentenze:

```
# apertura del file in cui verranno salvati i risultati
with open(output_file_path, 'w', encoding='utf-8') as f:
    # utilizzo di tqdm per la barra di completamento
    for i, amendment_file in enumerate(tqdm(valid_amendments_files,
    ↪  desc="Emendamenti", unit="emendamento")):
        amendment_vector = amendments_vectors[i]

        # controllo della validità del vettore
        if amendment_vector.size == 0:
            print(f"Emendamento vuoto ignorato: {amendment_file}")
            continue
        # reshape per rendere la forma del vettore compatibile con
        ↪  "cosine_similarity"
        amendment_vector = amendment_vector.reshape(1, -1)

        # calcolo della similarità tra l'emendamento corrente e
        ↪  tutte le sentenze
        for j, sentence_file in enumerate(sentences_vector_files):
            sentence_vector = sentences_vectors[j]

            # controllo della validità del vettore
            if sentence_vector.size == 0:
                print(f"Sentenza vuota ignorata: {sentence_file}")
                continue
```

```
22
23              sentence_vector = sentence_vector.reshape(1, -1)
24
25              # calcolo della similarità e scrittura sul file di
                ↪ output
26              similarity = cosine_similarity(amendment_vector,
                ↪ sentence_vector)[0, 0]
27              f.write(f"Similarità tra Emendamento
                ↪ '{extract_amendment_identifier(amendment_file)}' e
                ↪ {extract_sentence_identifier(sentence_file)}:
                ↪ {similarity:.4f}\n")
```

Il codice descritto produce in output un file di testo (.txt) con i risultati delle similarità. In particolare, in ogni riga sono elencati: un emendamento, una sentenza (con il relativo dispositivo) e il valore di similarità tra i due documenti. Di seguito è presente un estratto:

- Similarità tra Emendamento **AC1321/1_1.akn.xml** e Sentenza **1957-25** (Dispositivo: *illegittimità costituzionale*): **0.5123**
- Similarità tra Emendamento **AC1660/23_2.akn.xml** e Sentenza **2009-252** (Dispositivo: *inammissibilità*): **0.6282**
- Similarità tra Emendamento **AC1896/1_276.akn.xml** e Sentenza **2006-49** (Dispositivo: *non fondatezza*): **0.6901**
- Similarità tra Emendamento **AC1896/1_219.akn.xml** e Sentenza **2020-217** (Dispositivo: *illegittimità costituzionale*): **0.7031**
- Similarità tra Emendamento **AC1902/14_7.akn.xml** e Sentenza **1973-74** (Dispositivo: *non fondatezza*): **0.4949**
- Similarità tra Emendamento **AC1896/1_171.akn.xml** e Sentenza **1992-23** (Dispositivo: *illegittimità costituzionale*): **0.5332**

Per analizzare i dati ottenuti dal calcolo della similarità tra emendamenti e sentenze, vista l'imponente mole di informazioni (circa 15 milioni di confronti emendamento-sentenza), ho optato per trasformarli in un *DataFrame* Python usando la libreria

"pandas" [12], attraverso il quale poter manipolare più facilmente i dati e calcolare valori che migliorano l'analisi dei dati, come media, deviazione standard, coppie con similarità più alta o più bassa. Le colonne del *Dataframe* che ho creato includono: l'identificativo dell'emendamento, l'identificativo della sentenza (con indicato il dispositivo) e il valore di similarità. Dall'elaborazione di questi dati sono emersi dei dati statistici che descrivono la distribuzione delle similarità: sono stati calcolati 15'024'936 valori di similarità, la cui media è 0.5395, indicando che il livello di similarità tra emendamenti e sentenze è in media, moderato. La deviazione standard è pari a 0.0483: questo indica che la maggior parte dei valori è concentrato in modo abbastanza stretto intorno alla media (tra circa 0.4912 e 0.5878). Di seguito è allegato il grafico della distribuzione di similarità:

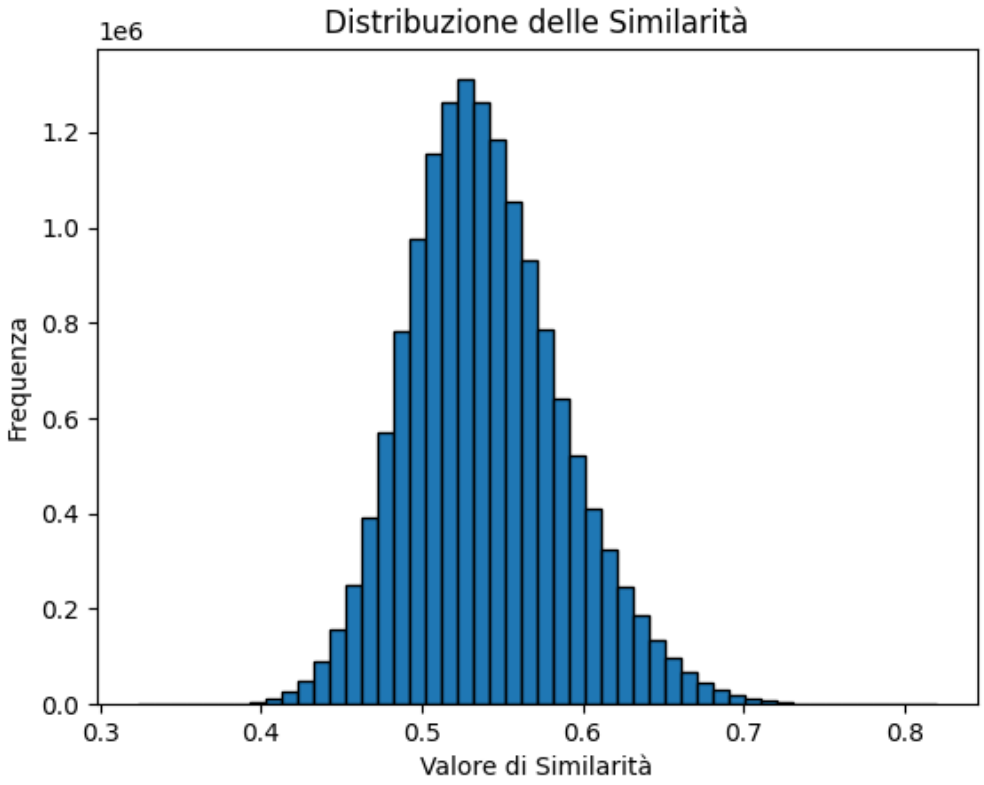

Figura 5.1: Distribuzione delle Similarità

Dall'analisi dei valori estremi sono emersi dei casi interessanti, sia con valori di simi-

larità molto alta, sia con valori molto bassi. La coppia formata dall'emendamento AC1896/2.01 e della sentenza 2018-166 presenta come valore di similarità 0.8053. Entrambi i documenti si concentrano su tematiche relative alle politiche abitative e alla regolamentazione delle condizioni per accedere a misure di sostegno legate all'alloggio. La sentenza affronta il tema dell'accesso alle misure di sostegno al canone di locazione, in particolare dichiara incostituzionale (violando il principio di uguaglianza sancito dall'articolo 3 della Costituzione) il requisito della residenza decennale nel territorio nazionale per i cittadini di paesi non appartenenti all'Unione Europea per ottenere le misure di sostegno. L'emendamento propone di creare un sistema per l'edilizia residenziale pubblica e sociale tramite disposizioni come l'individuazione dei criteri per assegnare gli alloggi, la realizzazione di un censimento del patrimonio edilizio utilizzato e l'implementazione di programmi di riqualificazione facendo attenzione alla sostenibilità sociale. Come conferma dell'alto valore di similarità, sono presenti importanti sinergie tra l'emendamento e la sentenza: entrambi i documenti trattano il tema del diritto all'abitazione come un diritto sociale fondamentale; l'emendamento propone misure per ridurre il disagio abitativo e garantire un'adeguata distribuzione degli alloggi sociali; la sentenza sottolinea il principio di uguaglianza ed elimina le discriminazioni che rendono difficile l'accesso alle misure di sostegno abitativo per soggetti vulnerabili come i migranti. Un caso di similarità molto bassa tra emendamento e sentenza che ho analizzato nel dettaglio è quello dell'emendamento AC1902-6.4 e della sentenza 1973-162, le quali presentano come valore di similarità 0.3384. L'emendamento tratta del tema dell'istruzione, in particolare sul potenziamento dei percorsi di formazioni per i docenti che offrono sostegno didattico agli alunni con disabilità. Vengono modificati i requisiti formativi introducendo laboratori pratici e sono aumentati i crediti formativi (CFU) da acquisire sia per iniziare a lavorare, sia negli anni successivi all'ottenimento del ruolo. La sentenza, d'altra parte, affronta una questione completamente diversa: viene trattato il tema della legittimità costituzionale di alcuni articoli del regio decreto del 1933, i quali disciplinano l'assegno bancario e circolare e del regio decreto del 1942, che riguardano alcune leggi sul fallimento e le procedure correlate (concordato preventivo e liquidazione). Da questo si evince e viene confermato dal valore di similarità, che i due documenti non trattano temi correlati né dal punto di vista semantico né a livello tematico.

Per concludere, la variabilità della similarità tra sentenze ed emendamenti è determinata dalla natura tematica e semantica dei documenti: la distribuzione dei valori di similarità è concentrata intorno a un valore medio di 0.5395 e sono presenti alcuni casi di valori estremi che ho analizzato, da cui ho tratto delle riflessioni e ho confermato i relativi valori di similarità.

Capitolo 6

Utilizzi, sviluppi futuri e conclusioni

6.1 Possibili utilizzi

Il sistema di analisi della similarità tra emendamenti e decisioni della Corte costituzionale descritto in questa tesi presenta numerosi potenziali utilizzi, sia in contesto giuridico, sia in ambiti più generali. Primo tra tutti, il supporto alle attività legislative: questo sistema può supportare i legislatori a identificare automaticamente gli emendamenti che convergono (o divergono) con i precedenti giuridici della Corte, permettendo di individuare quelli potenzialmente incostituzionali. Il sistema potrebbe anche essere esteso per verificare la coerenza tra emendamenti e leggi esistenti, non limitandosi alle decisioni della Corte, riducendo il rischio di conflitti tra leggi e potenziali situazioni di incostituzionalità. In generale, il sistema permetterebbe di ridurre il carico di lavoro nelle istituzioni legislative e giudiziarie, permettendo di dedicare più risorse ad altre attività. Professionisti del settore giuridico potrebbero utilizzare il sistema per confrontare emendamenti, leggi e sentenze, fornendo loro una migliore comprensione delle connessioni tra i documenti legislativi. In ambito accademico, il sistema può essere utilizzato come strumento per studiare le evoluzioni delle norme giuridiche e i loro legami con quelle precedenti, anche appartenenti a contesti internazionali, vista l'integrazione multilingua di bge-m3.

6.2 Miglioramenti futuri

Uno degli sviluppi più promettenti per migliorare il processo di embedding adottato nel progetto consiste nell'integrazione di metodologie basate su *"multi-layered embedding-based retrieval"*, come descritto nel paper di João Alberto de Oliveira Lima [5]. Questo approccio consente di rappresentare i testi giuridici attraverso diversi livelli di granularità, permettendo un'analisi più dettagliata delle strutture semantiche dei documenti normativi. Di seguito, sono descritti potenziali miglioramenti del sistema legati ai concetti di *multi-layered retrieval*, *aboutness* e *semantic chunking*. Il metodo *multi-layered* si contraddistingue per la capacità di rappresentare i testi non solo come blocchi unici, ma anche tramite componenti strutturali dei testi; per esempio, permette di creare *embedding* per livelli come articoli, sezioni o paragrafi. Col mio attuale approccio, gli *embedding* sono calcolati a livello di intero documento; implementare un sistema *multi-layered* rende possibile creare *embedding* separati per i diversi livelli gerarchici, con un conseguente miglioramento della precisione del sistema di *retrieval*. Un'altra tecnica introdotta nel paper è quella che gli autori chiamano *"aboutness"*, la quale permette di rappresentare ogni segmento di un testo tramite il tema principale di cui tratta. Il sistema che io ho implementato utilizza *embedding* di interi testi; integrando l'*aboutness* sarebbe possibile rappresentare in modo più preciso ciascun livello gerarchico (documenti, articoli, paragrafi...): ogni segmento di emendamento o decisione avrebbe un *embedding* che rappresenta il suo significato centrale e il loro confronto si concentrerebbe sulle parti tematicamente simili (grazie all'*aboutness*), riducendo il rumore e migliorando la precisione del *retrieval*. Un altro concetto per migliorare la rappresentazione semantica è il *"semantic chunking"*, che consiste nel dividere i documenti in segmenti più piccoli e semanticamente coerenti: ognuno rappresenta un elemento tematico unico, garantendo una maggiore granularità e precisione nella rappresentazione semantica. Si differenzia dal *chunking* tradizionale poiché quest'ultimo consiste nella creazione di semplici segmenti più piccoli, senza la caratteristica della coerenza tematica. Infine, un ultimo miglioramento al sistema, consiste nell'implementazione di un meccanismo di riferimenti per le sentenze che permetta al modello di IA di conoscere e analizzare i documenti a cui esse fanno riferimento. Per quanto riguarda gli emendamenti, è fornito, all'interno del contenuto combinato, l'articolo che viene emendato; aggiungere anche alle sentenze le parti degli articoli a cui viene fatto riferimento, permette

di eseguire un'analisi più approfondita.

Le proposte di miglioramento elencate consentiranno un perfezionamento del processo di *embedding* e dell'analisi semantica: integrando tecniche come l'*aboutness*, il *semantic chunking*, l'approccio *multi-layered* e migliorando il sistema di riferimenti per le sentenze, sarà possibile aumentare la precisione e l'affidabilità del sistema.

6.3 Conclusioni

All'interno del presente lavoro ho analizzato la similarità semantica tra gli emendamenti proposti presso la Camera dei deputati e le decisioni della Corte costituzionale, utilizzando tecniche avanzate di intelligenza artificiale e sfruttando lo standard Akoma Ntoso per la rappresentazione strutturata dei documenti legali. Attraverso un processo di *scraping* e la successiva conversione in formato Akoma Ntoso, ho creato un dataset strutturato di emendamenti. Parallelamente, ho estratto le sezioni rilevanti delle sentenze della Corte costituzionale per effettuare un confronto significativo tra i documenti. Grazie all'utilizzo del modello AI bge-m3, e della libreria FlagEmbedding, ho generato vettori densi rappresentativi dei testi, che mi hanno permesso di condurre un'analisi semantica approfondita. Il calcolo del coseno di similarità ha evidenziato una distribuzione dei valori concentrata intorno a una media di 0.5395, indicando un livello medio moderato di similarità tra emendamenti e sentenze e l'analisi dei casi con valori estremi ha confermato la coerenza tra i risultati ottenuti e il contenuto semantico dei documenti esaminati. Questo progetto ha dimostrato l'efficacia delle tecniche di intelligenza artificiale nel supportare l'analisi legislativa, offrendo strumenti che possono facilitare l'individuazione di convergenze o divergenze tra giurisprudenza consolidata e proposte legislative o addirittura l'incostituzionalità di quest'ultime. Tuttavia, esistono opportunità per migliorare ulteriormente il processo di *embedding*, ad esempio ottimizzandolo per catturare in modo più accurato le sfumature semantiche dei testi legali.

In conclusione, il lavoro svolto rappresenta un passo significativo verso l'integrazione dell'intelligenza artificiale nel processo legislativo: apre la strada a ulteriori sviluppi e applicazioni future che potranno migliorare l'efficienza e la qualità delle attività legislative.

Bibliografia

[1] M. Palmirani, F. Vitali, W. Van Puymbroeck, and F. N. Durango, *Legal Drafting in the Era of Artificial Intelligence and Digitisation*. Directorate-General for Informatics, European Commission, Brussels, Belgium, 2021.

[2] M. Palmirani, *Hybrid AI to Support the Implementation of the European Directive*. CIRSFID, University of Bologna, Italy, in *EGOVIS 2022, LNCS 13429*, 2022.

[3] M. Palmirani, F. Vitali, G. Longo, E. Di Sante, A. Brega, A. D'Arpa, and M. Corazza, "Legal Drafting Supported by AI: Enhancing LEOS," in *Ital-AI 2024: 4th National Conference on Artificial Intelligence*, Naples, Italy, May 2024.

[4] M. Palmirani, *Semantic Categorisation of Judicial Decisions in the Case Law Databases With Recommendations*, Council of Europe, Strasbourg, France, 2024.

[5] J. A. de Oliveira Lima, "Unlocking Legal Knowledge with Multi-Layered Embedding-Based Retrieval," University of Brasília; Federal Senate of Brazil, Nov. 2024. [Online]. Available: `https://arxiv.org/abs/2411.07739`.

[6] J. Chen et al., "M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation," arXiv, Jun. 2024. [Online]. Available: `https://arxiv.org/abs/2402.03216`.

[7] Hugging Face, *BAIA/bge-m3 Documentation*, 2024. [Online]. Available: `https://huggingface.co/BAAI/bge-m3`.

[8] Beautiful Soup, *Beautiful Soup Documentation*, 2024. [Online]. Available: `https://beautiful-soup-4.readthedocs.io/en/latest/`.

[9] Saxonica, *Saxon XSLT Processor Documentation*, 2024. [Online]. Available: https://www.saxonica.com/html/documentation12/about/index.html.

[10] Scikit-learn, *Cosine_Similarity Documentation*, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html.

[11] FlagEmbedding, "FlagEmbedding: A library for advanced text embeddings," 2024. [Online]. Available: https://pypi.org/project/FlagEmbedding/1.0.6/.

[12] Pandas Documentation, "Pandas: Python Data Analysis Library," 2024. [Online]. Available: https://pandas.pydata.org/docs/.

[13] Lxml Documentation, "lxml - Processing XML and HTML with Python," 2024. [Online]. Available: https://lxml.de/index.html#documentation.

[14] B. Trancón y Widemann and M. Lepper, *Simple and Effective Relation-Based Approaches to XPath and XSLT Type Checking*, Technical Report, Bad Honnef, 2015. [Online]. Available: http://arxiv.org/abs/1905.07362v1.