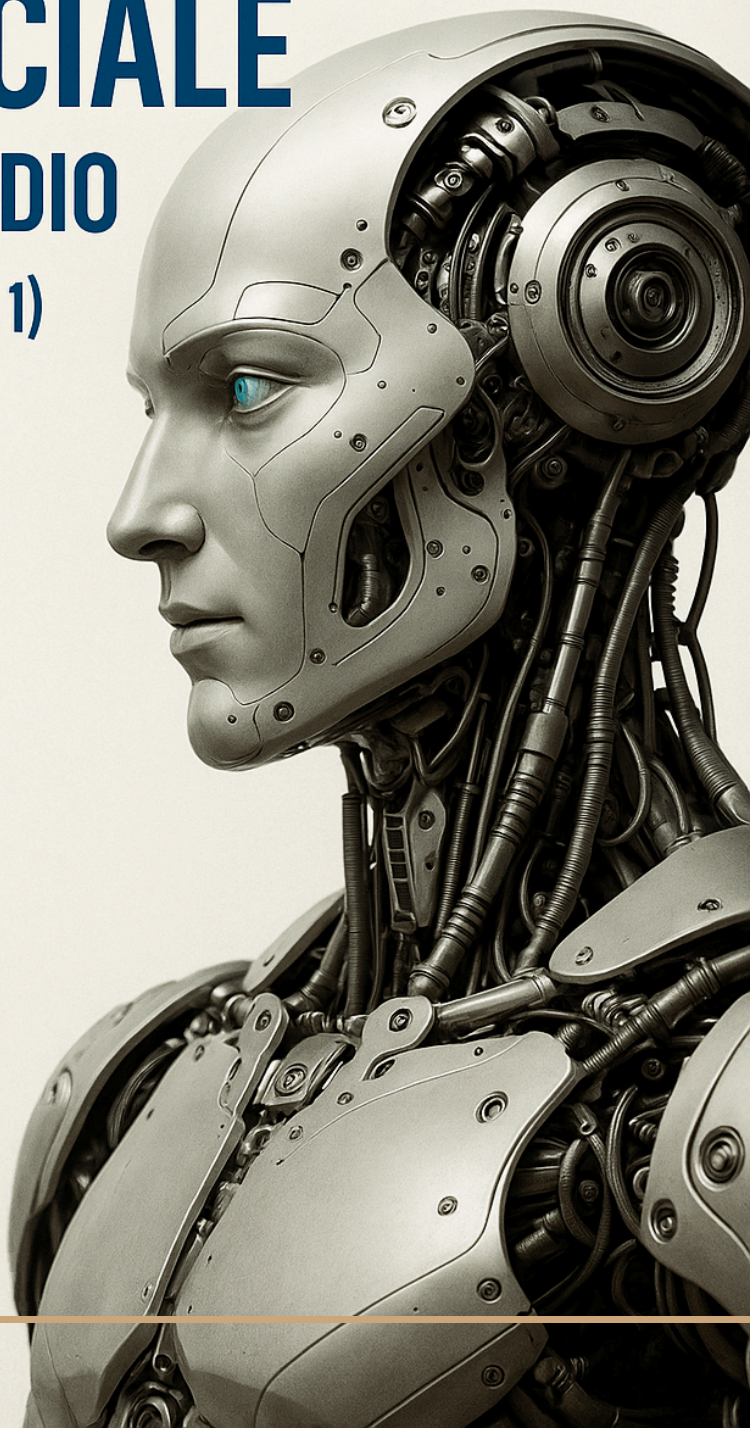
SPERIMENTARE L'INTELLIGENZA ARTIFICIALE

NELLO STUDIO LEGALE (Vol. 1)

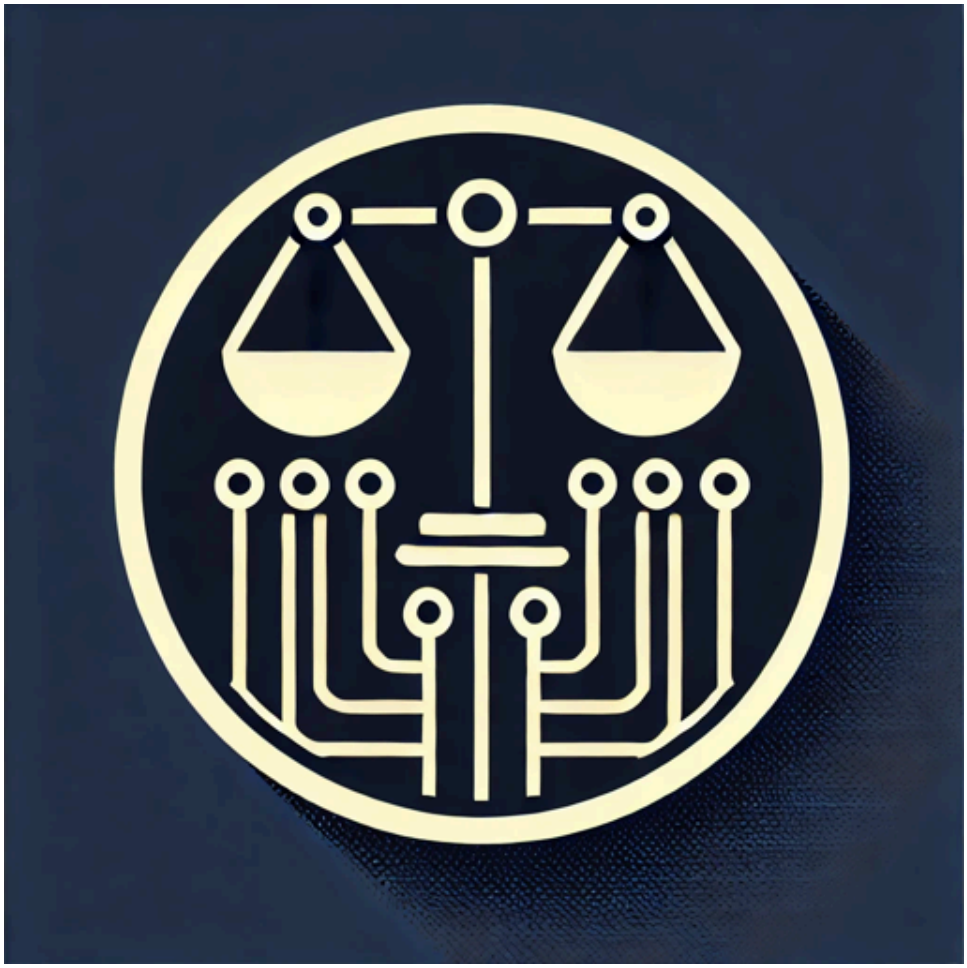


AVV.
GIUSEPPE
BRIGANTI

SPERIMENTARE L'INTELLIGENZA ARTIFICIALE NELLO STUDIO LEGALE (Vol. 1)

Modelli locali, codice commentato e casi d'uso reali per automazione e riservatezza

AVV. GIUSEPPE BRIGANTI



E-mail: contatti@iusreporter.it

Testi e immagini realizzati con l'assistenza del GPT Iusreporter di ChatGPT. Per informazioni: www.iusreporter.it, ricerca giuridica online dal 2001

Sommario

📄 Disclaimer

Gli script Python e i contenuti di questo ebook sono forniti esclusivamente a scopo informativo, didattico e sperimentale, e devono essere utilizzati unicamente in ambienti di test controllati. Non costituiscono consulenza legale o tecnica e non sostituiscono il parere di un professionista qualificato.

L'autore declina ogni responsabilità per eventuali errori, omissioni, malfunzionamenti, danni diretti o indiretti, incidentali, consequenziali o di altro tipo derivanti dall'uso, dall'uso improprio o dall'impossibilità di utilizzo degli script o delle informazioni contenute nel presente testo. Gli script sono forniti "così come sono" (AS IS), senza garanzie esplicite o implicite, incluse, ma non limitate a, garanzie di commerciabilità, idoneità a uno scopo particolare o assenza di violazioni.

L'utilizzo degli strumenti rimane sotto la piena responsabilità dell'utente, che è tenuto a verificarne, in particolare:

- *la correttezza tecnica e funzionale,*
- *la conformità alle normative vigenti, incluse, a titolo esemplificativo, l'AI Act, il GDPR, il Codice deontologico forense,*
- *il rispetto delle licenze delle librerie e dei componenti e software di terze parti eventualmente utilizzati, inclusi quelli distribuiti con licenze open source.*

Gli script non sono progettati né validati per l'uso in ambienti produttivi o per l'elaborazione di dati personali. Qualsiasi utilizzo in tali contesti è esclusiva responsabilità dell'utente, che deve adottare le opportune misure di sicurezza e valutare l'impatto normativo.

Gli script sono stati testati su Python 3.10 in ambiente Windows. Non è garantita la compatibilità con versioni diverse del linguaggio o con altri sistemi operativi. L'autore non fornisce assistenza tecnica, aggiornamenti periodici o correzione dei malfunzionamenti.

Tutti i contenuti di questo ebook, nessuno escluso, sono protetti dal diritto d'autore ai sensi della Legge 22 aprile 1941, n. 633 e successive modificazioni. È vietata la riproduzione, distribuzione, pubblicazione, comunicazione o modifica, totale o parziale, in qualsiasi forma, dei contenuti senza autorizzazione scritta dell'autore.

L'uso di questo testo e degli script in esso contenuti implica l'accettazione integrale del presente disclaimer. *Se non si accettano tali condizioni, si prega di non utilizzare il materiale fornito.*

Eventuali errori, osservazioni o segnalazioni possono essere comunicati all'autore, che si riserva di valutarli senza alcun obbligo di risposta o correzione.

Aggiornato al maggio 2025

Introduzione

Intelligenza artificiale generativa per studi legali: una breve guida per capire e sperimentare

L'innovazione tecnologica ha trasformato radicalmente molti settori, e l'ambito legale italiano non fa eccezione.

Negli ultimi anni, l'introduzione dell'intelligenza artificiale (IA), specialmente nella forma di modelli generativi (LLM – Large Language Models), ha aperto nuove prospettive e opportunità per Avvocati e studi legali.

Con questo ebook ci proponiamo di accompagnare gli Avvocati nella comprensione teorica e nell'implementazione pratica dell'IA generativa, focalizzandoci sulla tecnologia RAG (Retrieval-Augmented Generation) e altre applicazioni innovative.

Il testo raccoglie una serie di post pubblicati nel blog dell'Autore; i riferimenti a "post" s'intendono pertanto in questa sede rivolti ai capitoli del presente ebook.

Cos'è l'intelligenza artificiale generativa e perché interessa gli Avvocati?

Un modello generativo, noto anche come LLM, è un sistema che, partendo da un grande volume di dati testuali, è in grado di generare contenuti coerenti, rispondere a domande, sintetizzare documenti complessi e assistere nella ricerca giuridica.

Gli studi legali possono beneficiare enormemente di queste capacità, ottimizzando il lavoro, riducendo i tempi di elaborazione delle pratiche e migliorando la qualità del servizio offerto ai Clienti.

Tuttavia, l'applicazione degli LLM in ambito legale presenta sfide specifiche. Per esempio, la maggior parte dei modelli disponibili sono addestrati prevalentemente su dati in lingua inglese e sul sistema giuridico di *common law*.

Perché un sistema RAG?

La tecnologia RAG combina l'efficacia dei modelli generativi con un potente sistema di recupero delle informazioni da banche dati interne, consentendo risposte estremamente precise e fondate su fonti autorevoli e interne allo studio.

Questo approccio mitiga significativamente alcuni dei limiti più critici degli LLM tradizionali, come il rischio di "allucinazioni" (risposte errate generate con apparente sicurezza) e l'imprecisione nelle informazioni fornite.

Utilizzando strumenti locali come Ollama e Python è possibile implementare soluzioni che rispettano pienamente i criteri di riservatezza e sicurezza, fondamentali nel contesto legale.

Struttura della guida

Volume 1

Parte prima. Fondamenti teorici e contesto legale

- Introduzione all'IA generativa negli studi legali italiani.
- Il sistema RAG e il suo valore aggiunto per gli Avvocati.
- Contesto normativo essenziale: AI Act, GDPR e deontologia professionale.
- Scelta e sicurezza di architetture AI locali.
- Gli LLM.

Parte seconda. Guida pratica per sperimentare

- Preparazione dell’ambiente di lavoro.
- Avvio e gestione locale di modelli LLM tramite Ollama.
- Tecniche avanzate di elaborazione di documenti legali (OCR, NLP).
- Implementazione pratica di un motore RAG sperimentale.
- Creazione e ottimizzazione di prompt specifici per studi legali.
- Metodologie di testing, debugging e miglioramento continuo.
- Possibili espansioni del sistema RAG locale con focus sull’automazione della gestione documentale.
- Giustizia predittiva: un esempio pratico di analisi predittiva delle controversie legali.

Volume 2 (di prossima pubblicazione)
Approfondimenti

A chi si rivolge la guida?

Questo percorso è dedicato agli Avvocati italiani interessati a sperimentare per comprendere e sfruttare al meglio le potenzialità dell’intelligenza artificiale generativa.

Ogni capitolo è strutturato per offrire una lettura agevole e comprensibile, con approfondimenti tecnici opzionali.

Una conoscenza di base del linguaggio di programmazione Python è consigliata ma non indispensabile.

Conclusione

Con questa guida, miriamo a fornire un quadro esaustivo e pratico su come l’IA possa essere un potente alleato per lo studio legale moderno, capace di migliorare significativamente l’efficienza, la qualità del servizio e la competitività dello studio sul mercato.

Senza mai dimenticare, naturalmente, che l'Avvocato umano rimarrà sempre insostituibile.

Nota

L'ebook contiene script Python di esempio.

Come spiegato nel testo, il codice dovrà essere utilizzato con un editor di codice compatibile con Python, rispettando in particolare le regole di indentazione del linguaggio.

Inviando un messaggio all'Autore è possibile ricevere tutti i file .py degli script Python contenuti nel testo.
E-mail: contatti@iusreporter.it

Per facilitare la lettura del testo e l'applicazione degli esempi, e per approfondire i temi dell'IA nel settore legale, e del legal tech in generale, è possibile utilizzare Iusreporter.tech, il GPT di ChatGPT dedicato specificamente al legal tech. Per informazioni: www.iusreporter.it.

PARTE PRIMA

Fondamenti teorici e contesto legale

1. Introduzione all’intelligenza artificiale generativa per lo studio legale

IA e Avvocati: il potenziale dell’intelligenza artificiale per la professione legale

Negli ultimi anni, l’intelligenza artificiale generativa ha trasformato profondamente numerosi settori, incluso quello legale.

Al centro di questa rivoluzione tecnologica troviamo i **Large Language Models (LLM)**, ovvero modelli linguistici di grandi dimensioni addestrati su vasti insiemi di dati testuali provenienti da fonti eterogenee.

Un LLM, come GPT, Mistral o LLaMA, è in grado di comprendere, interpretare e generare testo in linguaggio naturale con un alto grado di coerenza, offrendo così nuove opportunità per automatizzare compiti ripetitivi, migliorare la ricerca legale e supportare l’elaborazione di grandi volumi di documenti negli studi legali.

Per uno **studio legale**, comprendere il potenziale di questi strumenti non è più un’opzione, ma una necessità strategica.

Non si tratta semplicemente di aggiornarsi tecnologicamente, ma di dotarsi di strumenti avanzati capaci di aumentare la produttività, ridurre i tempi di lavorazione e migliorare sensibilmente la qualità del servizio legale offerto al Cliente.

Esempi d’uso dell’IA generativa in ambito legale

L’impiego concreto dell’intelligenza artificiale generativa nel diritto è già una realtà in molte giurisdizioni, e l’Italia sta rapidamente recuperando terreno.

Alcuni esempi pratici includono:

- **Redazione automatica di atti, pareri e contratti**: grazie all’uso di prompt mirati (*legal prompting*), un LLM può generare bozze giuridiche di qualità, pronte per la revisione e la personalizzazione da parte del (*insostituibile*) Legale.
- **Sintesi e comprensione di documenti complessi**: l’IA è capace di leggere un contratto, una sentenza o una normativa e produrre un riassunto strutturato evidenziando clausole principali, rischi e punti critici.
- **Ricerche giurisprudenziali e normative assistite**: abbinando un LLM a un sistema RAG locale, è possibile interrogare l’intero archivio documentale dello studio legale e ottenere risposte contestualizzate basate su fonti rilevanti.
- **Verifica della conformità normativa (*compliance*)**: l’IA può analizzare documenti aziendali o contrattuali e segnalare automaticamente eventuali incongruenze rispetto alla normativa vigente.
- **Preparazione di *memorandum* e *report* legali**: riducendo i tempi di elaborazione per attività standard, gli Avvocati possono dedicarsi ad analisi più strategiche e consulenze di valore.

IA e Avvocati: esigenze, limiti e opportunità per gli studi legali italiani

Gli studi legali italiani, in particolare quelli di piccole e medie dimensioni, si trovano oggi ad affrontare sfide rilevanti:

- **Gestione di archivi documentali in costante espansione**
- **Rispetto stringente della normativa GDPR sulla protezione dei dati personali**

- **Necessità di mantenere il controllo diretto su dati riservati**
- **Ricerca di soluzioni tecnologiche sostenibili, modulari e personalizzabili**

In questo scenario, l'adozione di una soluzione di intelligenza artificiale **in locale** (*on-premises*), eseguita su infrastruttura controllata dallo studio legale stesso, rappresenta una svolta decisiva.

Questo approccio garantisce la massima riservatezza, elimina la dipendenza da fornitori esterni e consente una configurazione su misura per le esigenze del *team* legale.

Per esempio, con l'adozione di LLM **open source**, pienamente compatibili con strumenti come **Ollama**, è possibile creare un'infrastruttura efficiente e sicura a costi contenuti.

Un sistema RAG in locale rappresenta la scelta più sicura e autonoma. Tuttavia, esistono alternative ibride e soluzioni verticali che possono offrire un buon compromesso tra efficienza e semplicità di implementazione.

Le **soluzioni ibride** combinano l'esecuzione locale delle funzioni più sensibili (come il trattamento di dati personali o l'archiviazione di fascicoli) con l'uso del *cloud* per operazioni non critiche, come l'elaborazione linguistica o l'accesso a banche dati aggiornate. Questa configurazione consente di bilanciare *privacy*, costi e scalabilità.

Le **soluzioni verticali**, invece, sono piattaforme già progettate per il settore legale, che integrano modelli linguistici, database giuridici e interfacce intuitive. Offrono rapidità di implementazione e funzionalità specifiche come l'analisi predittiva o il riassunto di atti, ideali per studi che desiderano beneficiare dell'IA senza sviluppare internamente il proprio sistema.

In breve:

- **RAG locale**

  Priorità: Sicurezza assoluta (GDPR, dati sensibili).

  Ideale per: Studi con documentazione altamente riservata o nicchie iper-specializzate.
- **Ibrido**

  Priorità: Bilanciare sicurezza e scalabilità.

  Esempio: **archiviazione locale** di contratti/pareri **+ cloud** per ricerche normative generiche.
- **Verticali**

  Priorità: Efficienza immediata senza sviluppo interno.

Tabella di sintesi:

Tipo	Vantaggi	Svantaggi
Locale	Massimo controllo dati	Costi IT elevati
Ibrido	Flessibilità operativa	Gestione integrata complessa
Verticale	Ottimizzato per il legal italiano	Personalizzazione limitata

Tabella di confronto:

Feature	RAG Locale	Ibrido	Verticale
Costi iniziali	Alto (hardware)	Medio	Basso
Flessibilità	Massima	Alta	Bassa
Manutenzione	Complessa	Media	Gestita dal fornitore
GDPR Compliance	Ottimale	Dipende da setup	Variabile

Scelta ottimale:

- **Piccoli studi**: verticali preconfigurati
- **Medio-grandi studi**: ibrido (*cloud* per ricerche + locale per dati sensibili)
- **Nuclei iper-specializzati**: RAG locale con modelli addestrati su documentazione interna.

Perché iniziare ora

Gli LLM non sono più appannaggio esclusivo dei giganti della tecnologia o degli studi internazionali con grandi budget.

Con strumenti oggi disponibili e facilmente implementabili – come, per esempio, **Ollama, LangChain, FAISS e Tesseract** – anche uno studio medio-piccolo può avviare un semplice sistema RAG locale sperimentale per testare le possibilità offerte dalla IA, in totale autonomia.

Iniziare oggi consente di:

- **Costruire un vantaggio competitivo** duraturo
- **Formare risorse interne** su strumenti avanzati e innovativi
- **Migliorare il servizio al Cliente** in termini di efficienza, accuratezza e tempestività
- **Offrire nuovi servizi a valore aggiunto**, come la consulenza supportata dall'analisi automatizzata dei documenti

Nei prossimi articoli tratteremo in dettaglio:

- Cos'è e come funziona un sistema RAG (Retrieval-Augmented Generation)
- I principali requisiti di riservatezza e principi deontologici da rispettare nell'uso dell'IA nello studio legale
- Le caratteristiche tecniche e i vantaggi dei principali strumenti open source

- Una guida operativa per costruire passo passo un semplice esempio di sistema RAG locale
- Una guida operativa per interagire al meglio con l'IA attraverso le migliori tecniche di *legal prompting*.

Potremo così comprendere come funziona davvero un sistema AI e sperimentare in prima persona le sue possibili applicazioni nel settore legale.

2. Cos’è un sistema RAG e perché è rilevante per gli Avvocati

Introduzione

Nel panorama in rapida evoluzione dell’Intelligenza Artificiale (IA), una delle tecnologie più promettenti per gli studi legali è il sistema RAG, acronimo di *Retrieval-Augmented Generation*.

Questa architettura, che unisce modelli linguistici di grandi dimensioni (LLM) a sistemi di recupero documentale, rappresenta uno strumento importante per gli Avvocati che desiderano integrare l’IA nel proprio lavoro quotidiano, mantenendo al contempo il controllo sui dati riservati.

Cos’è un sistema RAG

Un sistema RAG si compone di due elementi fondamentali:

1. **Retriever**: modulo che cerca e recupera documenti rilevanti da una base di conoscenza locale (es. database, file PDF, atti giudiziari). Utilizza **tecniche avanzate di ricerca semantica**, basate su “embedding vettoriali”, che gli permettono di cogliere non solo le parole esatte ma anche il significato del testo. In altre parole, trova *ciò che serve davvero*, anche se i termini non coincidono perfettamente;
2. **Generatore (LLM)**: modello linguistico che analizza i documenti selezionati dal retriever per generare risposte coerenti, accurate e contestualizzate rispetto alla richiesta.

A differenza degli LLM “puri”, aggiornati solo fino a una certa data, un sistema RAG può essere costantemente aggiornato semplicemente modificando la documentazione di riferimento.

Perché un sistema RAG è utile per uno studio legale

I vantaggi per un Avvocato sono numerosi:

- **Aggiornamento costante**: lavora sempre sull’ultima versione dei documenti dello studio legale;
- **Risposte su misura**: produce risultati basati sui documenti interni dello studio;
- **Tutela massima della riservatezza**: se eseguito in locale, non invia dati a server esterni, garantendo conformità al GDPR e tutela del segreto professionale.

Alcune applicazioni pratiche in ambito legale

1. Ricerca giurisprudenziale personalizzata

Un sistema RAG locale, alimentato da sentenze annotate, consente di:

- Individuare precedenti rilevanti;
- Riassumere i punti salienti di una decisione;
- Mettere in luce orientamenti contrastanti tra tribunali.

2. Sintesi automatica di documenti

È possibile:

- Ottenere un riassunto di un nuovo contratto o atto;
- Confrontare versioni differenti di uno stesso documento;
- Evidenziare clausole critiche (es. penali, vincoli, ecc.).

3. Redazione assistita di pareri legali

In risposta a domande su casi concreti o ipotetici, il sistema può:

- Generare un primo schema di parere con riferimenti interni;
- Suggerire normativa pertinente;
- Offrire un punto di partenza per la stesura finale.

L’integrazione con LLM locali

Il massimo potenziale del sistema RAG si raggiunge utilizzando modelli linguistici locali, eseguibili con strumenti come **Ollama**, installati direttamente su macchine dello studio:

- **Nessun costo ricorrente per API**;
- **Massima riservatezza** dei dati trattati;
- **Adattabilità** a diverse branche del diritto (civile, penale, immigrazione, ecc.).

Una scelta strategica per lo studio legale

L’adozione di un sistema RAG consente di:

- **Aumentare l’efficienza**: automatizzando analisi e compiti ripetitivi;
- **Migliorare l’accuratezza**: grazie a risposte basate su fonti pertinenti;
- **Conservare il controllo**: mantenendo tutti i dati all’interno dell’infrastruttura dello studio.

Questa tecnologia trasforma l’IA da semplice strumento generico ad **assistente giuridico intelligente**, su misura per le esigenze operative dello studio.

Conclusione

Il sistema RAG rappresenta uno strumento importante per l’innovazione legale.

Abbiamo già visto le differenze tra sistemi RAG e **soluzioni ibride e verticali per lo studio legale**, evidenziando i rispettivi pro e contro in base alle caratteristiche dello studio.

Nei prossimi post esploreremo le implicazioni **deontologiche e normative**, con particolare riferimento al **GDPR** e al **Codice deontologico forense**, e successivamente procederemo passo passo alla creazione di un semplice sistema RAG d’esempio per lo studio legale.

3. IA e Avvocati: privacy e deontologia nell’uso dell’IA nello studio legale

L’introduzione dell’intelligenza artificiale negli studi legali italiani richiede un’analisi approfondita in termini di privacy, sicurezza dei dati, conformità normativa e deontologia professionale.

Questo articolo esamina il contesto giuridico aggiornato, i vantaggi delle soluzioni in locale e le prassi operative raccomandate per garantire un uso responsabile e conforme degli strumenti IA.

IA e Avvocati: quadro normativo (GDPR, AI Act, Codice deontologico forense e linee guida)

L’IA, specialmente quando impiegata per attività decisionali in uno studio legale, è soggetta a una pluralità di regole, quali, in particolare:

- il **Regolamento (UE) 2016/679 (GDPR)**,
- l’**AI Act**,
- e il **Codice deontologico forense**.

Art. 22 GDPR: decisioni automatizzate

Il GDPR vieta decisioni basate unicamente su trattamenti automatizzati dei dati personali che abbiano effetti giuridici o significativi sull’interessato, salvo eccezioni esplicite.

È fondamentale che ogni utilizzo dell'IA preveda un idoneo controllo umano.

AI Act: classificazione dei rischi e obblighi

Il Regolamento europeo sull'intelligenza artificiale (AI Act, Regolamento (UE) 2024/1689), in vigore dal 1° agosto 2024, distingue i sistemi di IA per livello di rischio:

- **Rischio inaccettabile**: vietato;
- **Rischio alto**: soggetto a requisiti stringenti;
- **Rischio limitato**: obblighi di trasparenza;
- **Rischio minimo**: nessun obbligo particolare.

Gli strumenti di intelligenza artificiale in ambito legale possono risultare talvolta ad **alto rischio** e richiedono, principalmente:

- **Sistema di gestione della qualità**
  Il fornitore deve implementare un sistema per garantire che il ciclo di vita dell'IA sia controllato e conforme.
- **Documentazione tecnica**
  Deve contenere informazioni dettagliate su: scopo del sistema, design, addestramento, validazione, monitoraggio, rischi e misure di mitigazione.
- **Registrazione e tracciabilità**
  Deve essere possibile risalire a decisioni, input, modifiche, output e configurazioni (log obbligatori).
- **Trasparenza e informazioni all'utente**
  Il sistema deve essere comprensibile per chi lo utilizza, con istruzioni chiare e indicazioni sui suoi limiti.
- **Supervisione umana**
  È richiesto che l'essere umano possa intervenire, annullare, verificare o bloccare l'operato del sistema.

- **Precisione, robustezza e cybersecurity**
  Il sistema deve essere tecnicamente solido, funzionare in modo affidabile e resistere a manipolazioni o attacchi informatici.
- **Valutazione di conformità**
  Obbligo per il fornitore di dimostrare la conformità del sistema prima dell'immissione sul mercato o della messa in servizio.

Principio di *accountability*

Il principio di *accountability* (art. 5.2 e 24 GDPR) impone allo studio legale di dimostrare, in qualsiasi momento, la propria conformità.

Le misure chiave includono:

- **policy interne documentate**;
- **valutazioni d'impatto sulla protezione dei dati (DPIA)**;
- **registro dei trattamenti**, anche per i flussi IA;
- **audit regolari** e tracciabilità delle decisioni;
- **log e documentazione** delle interazioni IA.

Codice deontologico forense

L'art. 13 impone la **riservatezza** anche per gli strumenti digitali.

L'art. 14 impone la **competenza**: l'Avvocato deve comprendere il funzionamento dell'IA e i suoi limiti, pena una potenziale violazione disciplinare.

Linee guida FBE e Carta dei principi dell'Ordine degli Avvocati di Milano

La **Fédération des Barreaux d'Europe (FBE)** raccomanda, in particolare, con le sue linee guida per l'uso dell'IA nel settore legale:

- supervisione umana obbligatoria;
- trasparenza degli strumenti;
- mitigazione dei *bias*;
- protezione dei dati sensibili.

La **Carta dei principi dell'Ordine degli Avvocati di Milano** sottolinea che l'IA deve affiancare, non sostituire, il giudizio dell'Avvocato.

Questo principio sancisce la centralità insostituibile del professionista legale nel processo decisionale, ribadendo che l'Avvocato deve mantenere il controllo critico e la responsabilità delle proprie valutazioni, anche quando si avvale di strumenti tecnologici avanzati.

L'uso dell'IA deve quindi essere sempre subordinato alla supervisione e all'autonomia del professionista, che resta garante della qualità, della correttezza e dell'etica dell'attività.

La Carta richiama inoltre altri valori fondamentali, come la trasparenza, la non discriminazione e l'aggiornamento continuo, delineando un quadro etico che integra innovazione e tradizione forense, in linea con i principi europei di tutela dei diritti e delle libertà fondamentali.

Soluzioni IA in locale: privacy-by-design e auditabilità

Le soluzioni cloud possono comportare rischi legati a trasferimenti extra-UE dei dati personali, accessi non autorizzati e a perdita di controllo sui dati.

Una **soluzione *on-premises*** offre maggiore sicurezza e conformità.

Vantaggi dell'architettura locale

- Controllo diretto e integrale sui dati;
- Nessun trasferimento internazionale;
- Facilità nell'applicazione del principio di *privacy-by-design*;
- Strumenti *open source*, ispezionabili e modificabili.

Alcuni strumenti disponibili

- **Ollama**: esecuzione di LLM locali;
- **FAISS**: indicizzazione e ricerca semantica;
- **Tesseract OCR**: estrazione da PDF scannerizzati;
- **LangChain**: orchestrazione di flussi IA complessi.

Questi strumenti, combinati, permettono di costruire un semplice sistema RAG locale e sicuro per testare le possibilità dell'IA nello studio legale.

IA e Avvocati: prassi operative per l'uso etico dell'IA

Per assicurare conformità e tutela dei diritti, l'Avvocato è tenuto quindi, in particolare, a:

- Informare il Cliente sull'uso dell'IA;
- Richiedere un consenso esplicito per trattamenti automatizzati;

- Mantenere la supervisione umana;
- Tenere traccia delle attività IA;
- Aggiornarsi regolarmente su normativa e tecnologia;
- Applicare cifratura, segmentazione e backup sicuri;
- Stabilire e formalizzare procedure interne per l'utilizzo responsabile e conforme dell'IA nello studio legale;
- Valutare il rischio legale ed etico dell'uso di specifici strumenti IA (specie se generativi);
- Evitare piattaforme pubbliche non controllate.

IA e Avvocati: conclusioni

L'uso dell'IA nello studio legale può offrire efficienza e innovazione, ma solo nel rispetto della privacy, della normativa vigente e dei principi deontologici.

Le architetture locali e trasparenti, supportate da una documentazione accurata e un controllo umano costante, offrono un modello sicuro e professionale.

4. IA e Avvocati: sicurezza e riservatezza

Introduzione

Nel contesto degli studi legali italiani, la sicurezza dei dati e la riservatezza delle informazioni trattate rappresentano requisiti imprescindibili, sanciti in particolare dal Codice deontologico forense e dal Regolamento UE 2016/679 (GDPR).

L'adozione di soluzioni di intelligenza artificiale (IA) generativa deve pertanto rispettare tali principi fondamentali, garantendo il pieno controllo e la protezione dei dati personali.

Questo articolo approfondisce l'importanza di scegliere un'architettura IA che assicuri la riservatezza e la sicurezza delle informazioni sensibili trattate negli studi legali.

Controllo dei dati e indipendenza

Nel settore legale, la gestione di informazioni altamente sensibili richiede soluzioni che consentano il massimo controllo sui dati.

Come visto, si distinguono diverse tipologie di sistemi IA:

- **Sistemi basati su Retrieval-Augmented Generation (RAG) locali**, che eseguono tutto *in-house*;
- **Sistemi ibridi**, che combinano risorse locali e *cloud*;
- **Soluzioni verticali**, progettate specificamente per ambiti giuridici o settori particolari.

Le architetture IA locali offrono vantaggi fondamentali:

- **Controllo totale sui dati:** i documenti rimangono all'interno dello studio, riducendo il rischio di accessi non autorizzati;
- **Conformità al GDPR:** nessun trasferimento verso server esteri, garantendo il rispetto delle normative europee;
- **Indipendenza tecnologica:** eliminazione della dipendenza da fornitori terzi, con maggiore autonomia operativa e personalizzazione.

L'utilizzo di modelli IA *open source* eseguiti localmente rappresenta una soluzione sicura ed efficiente per gli studi legali, pur comportando alcune sfide quali costi di *setup*, manutenzione e necessità di competenze tecniche dedicate.

Tali aspetti vanno valutati attentamente rispetto alle alternative ibride o verticali, che possono offrire un buon compromesso tra sicurezza e praticità.

Strumenti *open source* e approccio *privacy-by-design*

Un'architettura IA locale si basa su strumenti progettati secondo i principi della ***privacy-by-design***, ovvero concepiti fin dall'inizio per garantire la massima protezione dei dati personali.

Tra i principali strumenti *open source* che saranno utilizzati per implementare un semplice sistema RAG locale di esempio, supportati da ampie comunità di sviluppo, vi sono:

- **Ollama**
  Consente l'esecuzione locale di modelli linguistici avanzati, come Mistral e LLaMA.
  Disponibile nativamente per Windows, supporta GPU NVIDIA e AMD, e può funzionare completamente offline, assicurando l'isolamento dei dati sensibili.

- **FAISS (Facebook AI Similarity Search)**
  Libreria ad alte prestazioni per la gestione di database vettoriali, fondamentale per l'indicizzazione semantica e la ricerca efficiente nei documenti legali.
- **Tesseract OCR**
  Motore OCR sviluppato da Google, utilizzato per convertire PDF scannerizzati in testo leggibile e indicizzabile.
- **LangChain**
  Framework Python per la creazione di flussi di lavoro complessi basati su modelli di linguaggio (LLM). Connette Ollama con basi documentali indicizzate da FAISS, consentendo *prompt* avanzati e catene logiche.

Vantaggi specifici per gli studi legali

I benefici concreti di un'infrastruttura IA locale per un professionista del diritto includono:

- **Tutela del segreto professionale**, in linea con il Codice deontologico forense;
- **Aderenza al GDPR**, evitando la trasmissione di dati personali fuori dall'Unione Europea;
- **Controllo e personalizzazione**, con possibilità di adattare il sistema alle specificità dello studio;
- **Ottimizzazione dei costi**, grazie alla riduzione delle spese ricorrenti tipiche dei servizi *cloud*.

Va tuttavia considerato che soluzioni ibride o *cloud*, se correttamente configurate, con server ubicati in UE e misure di sicurezza adeguate, possono anch'esse garantire conformità normativa e flessibilità operativa.

Requisiti tecnici minimi (aggiornati all'aprile 2025)

Per implementare un semplice sistema RAG locale d'esempio, stabile e performante, si raccomandano le seguenti specifiche hardware e software:

- **CPU:** processore multicore moderno;
- **RAM:** 16 GB per LLM fino a 7 miliardi di parametri; 32 GB per modelli da circa 13 miliardi di parametri; 64 GB per modelli avanzati da 70 miliardi di parametri o superiori
  Nota: l'uso di modelli quantizzati o distillati può ridurre i requisiti di memoria;
- **GPU (opzionale ma consigliata):** NVIDIA RTX 3060 o superiore (8–12 GB VRAM), con *driver* aggiornati e supporto CUDA (utile in ambienti WSL2); Supporto crescente per GPU AMD con ROCm;
- **Sistema operativo:** Windows 10/11 (Ollama nativo) oppure distribuzioni Linux; WSL2 necessario solo per *tool* Linux specifici o *container* Docker;
- ***Storage*:** SSD con almeno 20–40 GB di spazio libero per modelli, documenti e cache;
- **Sicurezza:** implementare *backup* regolari, *firewall*, antivirus e aggiornamenti costanti per proteggere i dati e il sistema.

Conclusione

L'adozione di un'architettura IA locale, se adeguatamente dimensionata e supportata, rappresenta una scelta lungimirante per gli studi legali italiani, offrendo sicurezza, autonomia, rispetto delle normative e piena adattabilità alle esigenze specifiche.

Le tecnologie *open source* oggi disponibili permettono di costruire soluzioni robuste anche senza competenze tecniche avanzate, grazie alla trasparenza degli strumenti e al supporto di comunità specializzate.

Nei prossimi articoli verranno approfondite le fasi operative di installazione, configurazione e utilizzo concreto di un semplicissimo sistema RAG locale di esempio, con particolare attenzione al contesto giuridico italiano.

5. IA e Avvocati: LLM *open source* per studi legali

L'avvento dei modelli linguistici di grandi dimensioni (Large Language Models, o LLM) ha aperto prospettive radicalmente nuove per l'automazione delle attività legali.

Per gli studi legali italiani che desiderano adottare l'IA in modo responsabile e indipendente, la scelta di LLM *open source* si rivela una strada particolarmente interessante.

Questi strumenti, sviluppati da comunità scientifiche e aziende tecnologiche con approccio *open*, consentono una maggiore personalizzazione e aderenza alle esigenze specifiche dello studio.

Questo articolo fornisce una panoramica aggiornata su alcuni LLM "leggeri" e utili nel contesto legale, valutandone caratteristiche tecniche, compatibilità con Ollama e implicazioni etiche, oltre a offrire considerazioni pratiche per un'adozione consapevole.

Perché scegliere LLM *open source*?

I vantaggi degli LLM *open source* sono molteplici e si rivelano strategici per lo studio legale moderno:

- **Controllo completo dei dati**: esecuzione interamente in locale, a tutela della privacy, del segreto professionale e della deontologia.
- **Flessibilità**: possibilità di personalizzazione per ambiti specifici come diritto civile, penale, amministrativo, ecc.
- **Trasparenza**: accesso al codice e ai pesi del modello, evitando logiche opache o non verificabili.
- **Sostenibilità economica**: eliminazione di costi di licenza e dipendenza da piattaforme *cloud*.

Questi elementi si allineano perfettamente con le esigenze di uno studio legale che opera nel rispetto del GDPR e del Codice deontologico forense.

D'altra parte occorre, come già illustrato, tenere in considerazione le caratteristiche del singolo studio legale, scegliendo tra sistemi in locale, ibridi (locale + *cloud*) o soluzioni verticali.

LLM *open source*: quali modelli considerare oggi?

Tenendo presente l'obiettivo di implementare un semplice sistema RAG locale d'esempio, concentriamoci su due dei modelli più efficaci e attivamente sviluppati nel 2025: **Mistral 7B**, un vero modello *open source*, e **LLaMA 3.1 8B**, un modello a codice accessibile ma con licenza d'uso ristretta.

Mistral 7B

- **Punti di forza**: equilibrio ideale tra velocità di esecuzione, leggerezza e capacità di ragionamento logico.
- **Perfetto per**: redazione di bozze, sintesi di testi normativi, supporto nella ricerca interna.
- **Compatibilità**: pienamente supportato da Ollama, anche su hardware non specialistico.

LLaMA 3.1 8B (Meta)

- **Punti di forza**: maggiore accuratezza e coerenza nella generazione del linguaggio rispetto alla versione precedente.
- **Licenza**: rilasciato sotto la **Meta Llama 3 Community License**. L'uso in ambito professionale interno (es. studio legale) è permesso senza restrizioni. Per aziende/gruppi societari con oltre 700 milioni di utenti attivi mensili, è

obbligatorio richiedere un'autorizzazione esplicita a Meta per qualsiasi utilizzo (incluso SaaS).

- **Compatibilità**: pienamente integrabile con Ollama, anche via terminale o script python.

Tabella di confronto tra Mistral e LLaMA

Modello	RAM consigliata	VRAM consigliata	Compatibilità con Ollama	Note principali
Mistral 7B	16 GB	8-12 GB	Sì	Leggero e veloce, adatto a studi medi
LLaMA 3.1 8B	16-32 GB	12-16 GB	Sì	Maggiore accuratezza, richiede più risorse

Entrambi i modelli funzionano su Windows, macOS e Linux. Supportano l'integrazione via API per sistemi documentali, chatbot, motori di ricerca interni e strumenti di sintesi.

Bias e considerazioni etiche

I modelli linguistici di grandi dimensioni (LLM), pur essendo potenti strumenti di supporto, presentano limiti e rischi etici che è fondamentale conoscere e gestire con attenzione, soprattutto in ambito giuridico.

Bias culturali, linguistici e di genere

Molti LLM sono addestrati principalmente su dati in lingua inglese e su fonti generaliste, con una predominanza di contenuti provenienti da contesti culturali anglofoni. Questo comporta:

- ***Bias* linguistici e culturali:** difficoltà nella gestione accurata di lingue diverse dall'inglese, come l'italiano, e nella comprensione delle specificità culturali e normative locali.
- ***Bias* di genere:** ad esempio, l'uso del maschile sovraesteso nei dati di addestramento può portare a una rappresentazione non inclusiva, con conseguenze cognitive e sociali rilevanti. Le linee guida linguistiche italiane suggeriscono strategie come lo sdoppiamento (maschile e femminile affiancati) per mitigare questi effetti, ma la sfida resta aperta.
- ***Bias* contro categorie vulnerabili:** i modelli possono riflettere e amplificare stereotipi o pregiudizi contro disabili, minoranze o altre categorie protette, generando *output* discriminatori o offensivi.

Allucinazioni e affidabilità delle informazioni

Gli LLM possono produrre risposte plausibili ma errate o fuorvianti, fenomeno noto come "allucinazioni".

Questo rischio è particolarmente critico in ambito legale, dove l'accuratezza e la correttezza delle informazioni sono imprescindibili.

Opacità e trasparenza

Il funzionamento interno degli LLM è spesso opaco e non sempre prevedibile.

La mancanza di trasparenza può complicare la valutazione critica delle risposte generate e l'individuazione di eventuali errori o *bias*.

Strategie di mitigazione e responsabilità d’uso

Per un utilizzo responsabile degli LLM in ambito legale, è necessario adottare misure concrete:

- **Personalizzazione su *corpora* giuridici italiani:** addestrare o adattare i modelli su dati specifici del diritto italiano per migliorare la pertinenza e ridurre i *bias* linguistici e culturali.
- **Verifica umana obbligatoria:** integrare sempre una revisione da parte di professionisti qualificati nelle fasi critiche, per garantire correttezza e affidabilità.
- **Integrazione con motori di Retrieval-Augmented Generation (RAG):** combinare la generazione automatica con il recupero di fonti giuridiche certificate e aggiornate, aumentando la trasparenza e la tracciabilità delle informazioni.
- **Monitoraggio continuo e aggiornamento:** valutare periodicamente le prestazioni del modello e aggiornare i dati e le strategie di mitigazione in base all’evoluzione normativa e tecnologica.
- **Consapevolezza etica e giuridica:** rispettare i principi fondamentali di non discriminazione, tutela della privacy e trasparenza, in linea con le normative europee e italiane (es. AI Act, GDPR).

Questa visione integrata consente di sfruttare le potenzialità degli LLM mantenendo un controllo rigoroso sui rischi etici e giuridici, essenziale per garantire un uso responsabile e affidabile in ambito professionale.

Conclusioni

Per uno studio legale, adottare LLM come Mistral (*open source*) e LLaMA (codice accessibile con licenza dedicata) rappresenta una scelta innovativa e responsabile.

L'esecuzione in locale, la compatibilità con Ollama e la possibilità di adattamento rendono questi strumenti ideali per migliorare efficienza, qualità e riservatezza.

Nel prossimo articolo vedremo come preparare l'ambiente tecnico per implementare un semplicissimo sistema RAG d'esempio, con istruzioni pratiche per l'installazione e configurazione di Python, Ollama e FAISS, anche per chi non ha competenze di programmazione avanzata.

PARTE SECONDA

Guida pratica per sperimentare

6. Preparare un ambiente Windows per un sistema RAG locale in uno studio legale

Introduzione

Con questo sesto post inizia la guida pratica per l'implementazione di un semplice sistema RAG (Retrieval-Augmented Generation) locale di base all'interno di uno studio legale.

Un sistema RAG combina il recupero di documenti rilevanti con la generazione di testo tramite modelli di linguaggio di grandi dimensioni (LLM), permettendo di integrare l'intelligenza artificiale generativa nel flusso di lavoro legale.

L'obiettivo di questa fase è configurare correttamente un ambiente Windows da zero, garantendo stabilità, sicurezza e prestazioni ottimali per tutto il progetto. Questa base consente anche agli studi meno strutturati di avviare con successo l'adozione dell'IA generativa.

Requisiti minimi per il sistema RAG (aggiornati all'aprile 2025)

I requisiti hardware e software dipendono dal modello linguistico utilizzato e dal carico di lavoro previsto.

Per modelli leggeri (≤ 1 miliardo di parametri), è possibile operare con hardware limitato, ma per un uso stabile e produttivo in ambito legale si consiglia:

- **Sistema operativo:** Windows 11 Pro (consigliato per funzionalità avanzate di sicurezza) o Windows 10 versione 21H2 o superiore
- **Processore:** Intel i5 10ª generazione o AMD Ryzen 5 equivalente
- **RAM:** minimo 16 GB (preferibilmente 32 GB)
- **Spazio su disco:** almeno 30 GB liberi su SSD
- **GPU (opzionale):** NVIDIA con almeno 6 GB di VRAM e supporto CUDA (consigliata per accelerare l'elaborazione)

L'uso della GPU non è obbligatorio, ma riduce significativamente i tempi di elaborazione, soprattutto con modelli più complessi.

Installazione di Python

1. Scaricare Python (versione 3.11 o superiore) da: https://www.python.org/downloads/windows/
2. Durante l'installazione, selezionare "Add Python to PATH" per facilitare l'uso da terminale.
3. Verificare l'installazione aprendo il terminale (cmd o PowerShell) e digitando: python --version
4. In caso di problemi con il PATH, verificare manualmente la variabile d'ambiente.

Installazione di Visual Studio Code (VS Code)

1. Scaricare VS Code da: https://code.visualstudio.com/
2. Durante l'installazione, abilitare:

- "Aggiungi a PATH"
- "Apri con Code"

3. Dal marketplace di estensioni di VS Code, installare:

- Python
- Pylance (per supporto avanzato al linguaggio)

4. Configurare l'estensione Python per usare l'interprete dell'ambiente virtuale (.venv) che creeremo.

Installazione di Ollama per Windows

Ollama è un software che consente di eseguire LLM localmente. Dal 2024 è disponibile una versione nativa per Windows 10 (21H2) e Windows 11.

1. Visitare: https://ollama.com/
2. Scaricare e installare il file .exe.
3. Dopo l'installazione, aprire PowerShell o cmd e digitare, per scaricare e avviare il modello linguistico Mistral:

```
ollama run mistral
```

4. Questa operazione scarica e avvia un LLM pronto all'uso offline, con cui è possibile iniziare subito a interagire.

Nota: Verificare che firewall o antivirus non blocchino il download o l'esecuzione di Ollama.

Creazione dell'ambiente virtuale Python per il sistema RAG

1. Aprire una nuova cartella progetto (Python) in VS Code.
2. Aprire il terminale integrato (Visualizza > Terminale).

3. Creare l’ambiente virtuale:

```
python -m venv .venv
```

4. Attivare l’ambiente virtuale:

- In PowerShell (predefinito in VS Code su Windows):
  `.\.venv\Scripts\Activate.ps1`
  (Se l’esecuzione degli script è bloccata, sbloccarla con il comando appropriato)
- Nel Prompt dei comandi (cmd):
  `.venv\Scripts\activate.bat`

4. Aggiornare pip:

```
python -m pip install --upgrade pip
```

L’ambiente virtuale isola le librerie del progetto, evitando conflitti con altri progetti o con l’installazione globale di Python.

Consigli pratici per studi legali

Backup automatici

- Effettuare backup giornalieri automatici su disco esterno o NAS locale.
- Usare software affidabili e verificare periodicamente integrità e recuperabilità.
- Conservare almeno una copia cifrata in una posizione sicura e separata.

Sicurezza informatica

- Attivare BitLocker per la crittografia completa del disco.
- Mantenere aggiornati antivirus e firewall, preferendo suite integrate.
- Creare account separati per ogni collaboratore e gestire i permessi di accesso ai file.

- Applicare regolarmente aggiornamenti di sistema e software per chiudere vulnerabilità.
- Considerare l’uso di VPN o reti protette per accessi remoti.

Protezione dei dati personali

- Applicare rigorosamente le norme GDPR nella gestione dei dati.
- Definire policy chiare sull’uso dell’intelligenza artificiale e sulla gestione delle informazioni sensibili.

Gestione documentale

- Standardizzare i nomi delle cartelle (es. `2025_Rossi_vs_Comune_Roma`).
- Suddividere per cliente, anno e materia (es. `ricorsi/amministrativo/2025`).
- Usare strumenti di ricerca testuale per navigare rapidamente l’archivio digitale.
- Effettuare backup differenziati per cartelle operative e storiche.

Formazione interna

- Organizzare sessioni pratiche brevi (30-45 min) sull’uso del sistema.
- Coinvolgere i collaboratori nella fase di test e raccolta feedback.
- Designare un “referente digitale” interno allo studio.
- Fornire esempi reali e casi d’uso specifici (es. redazione di bozze di pareri o sintesi di sentenze).

Manuale operativo interno

Predisporre un documento PDF contenente:

- Guida passo-passo all’uso del sistema.

- Screenshot dei comandi principali (es. prompt in Ollama).
- Suggerimenti e scorciatoie utili.
- FAQ con risposte rapide ai problemi più comuni.
- Contatti interni o tecnici per assistenza.

Conclusione

Con l’ambiente Windows correttamente configurato, lo studio legale è pronto a sperimentare le potenzialità dell’intelligenza artificiale generativa locale.

Nel prossimo post vedremo nel dettaglio come scaricare, avviare e testare un LLM tramite Ollama, eseguendo i primi prompt giuridici.

7. IA e Avvocati: utilizzare un LLM locale con Ollama

Introduzione

In questo settimo passo della guida all'implementazione di un semplice sistema RAG di prova per studi legali, ci concentriamo sull'utilizzo pratico di un LLM locale (modello linguistico di grandi dimensioni) attraverso Ollama, scegliendo come esempio il modello LLaMA 3.1 8B.

L'obiettivo è dimostrare come un modello linguistico moderno possa essere utilizzato su macchina locale, garantendo sicurezza, indipendenza dal *cloud* e conformità normativa.

Dopo aver esplorato nel post precedente come installare Ollama e predisporre l'ambiente di lavoro su Windows, ora ci dedichiamo all'impiego reale di un LLM in uno scenario operativo.

Dopo aver predisposto l'ambiente tecnico e configurato Ollama, è fondamentale capire come interagire con un modello linguistico di ultima generazione, per poi sperimentare le sue potenzialità direttamente su documenti e tematiche giuridiche.

Questo passaggio costituisce un ponte tra la preparazione teorico-tecnica e l'uso concreto per applicazioni legali quotidiane, come la redazione assistita di pareri, l'analisi di testi normativi o la sintesi automatica di provvedimenti.

Avvio del modello LLaMA 3.1 8B con Ollama su Windows

Nel post precedente abbiamo visto come creare l'ambiente Windows necessario per utilizzare un LLM in locale con Ollama.

Comando per l'esecuzione (tramite terminale):

ollama run llama3.1:8b

Questo comando consente di:

- Scaricare automaticamente il modello (se non è già presente in locale)
- Eseguire il modello in modalità interattiva tramite terminale
- Avviare una sessione di *prompt* dove porre domande direttamente al modello in linguaggio naturale

Definizione di *prompt*

Prompt è il termine che indica la domanda, l'istruzione o il testo che si fornisce a un modello linguistico (LLM) per ottenere una risposta o un'elaborazione specifica.

In pratica, è il modo in cui "si parla" all'intelligenza artificiale, formulando richieste in linguaggio naturale o strutturato per guidare la generazione di contenuti.

Esempio di interazione con LLM locale:

> ollama run llama3.1:8b
>>> Quali sono i principi fondamentali della Costituzione italiana?

Premendo *Invio* il *prompt* sarà inviato al modello linguistico, che elaborerà la sua risposta.

Ecco un esempio di risposta (e dei limiti del modello):

La Costituzione Italiana è un documento di rilievo e importanza per il Paese. Essa stabilisce come vengono distribuite le competenze tra lo stato centrale ed enti locali, regola la rappresentanza parlamentare e stabilisce i principi fondamentali della Repubblica italiana.

I principi fondamentali sono l'unità, l'indivisibilità, la sovranità, la libertà, l'eguaglianza sociale.

Il modello LLaMA 3.1 8B è multilingue e si comporta bene anche in italiano. È particolarmente utile per:

- Creare riassunti automatici di provvedimenti
- Offrire spiegazioni comprensibili di articoli di legge o istituti giuridici
- Sviluppare bozze di pareri o relazioni partendo da contenuti normativi o giurisprudenziali
- Generare esempi pratici da utilizzare nella didattica o nella formazione interna

Pur essendo uno strumento potente, è sempre necessario validare le risposte fornite dal modello con la propria competenza professionale.

In una serie di post successiva sarà affrontato nel dettaglio il tema del *legal prompting*. Viene fornito intanto uno schema pratico per la formulazione di *prompt* efficaci.

Schema pratico per formulare domande efficaci

(Metodo R-CAFAR)

R – Ruolo

Definisci preliminarmente il **ruolo** che l'IA deve assumere o il punto di vista da adottare.

Esempio: "*Sei un avvocato esperto in diritto civile italiano.*"

C – Contesto

Specifica la **materia giuridica e la tipologia** della questione.

Esempio: "*Diritto delle locazioni – locazione abitativa – redazione di una diffida per morosità.*"

A – Attori e Fatti

Indica **chi è coinvolto** e **quali sono i fatti** rilevanti.

Esempio: "*Conduttore moroso da tre mesi (gennaio, febbraio, marzo 2025), contratto registrato a Roma.*"

F – Finalità

Definisci l'**obiettivo** della richiesta.

Esempio: "*Richiedo una bozza di diffida.*"

A – Aggiornamento

Specifica la necessità di **riferimenti normativi o giurisprudenziali aggiornati**.

Esempio: "*Normativa aggiornata al 2025.*"

R – Richieste specifiche

Dettaglia gli **elementi da includere** nella risposta, come riferimenti normativi, sintesi, bozze, e prevedi una **fase di verifica** per garantire accuratezza e completezza.

Esempio: "*Fornisci riferimenti normativi, bozza sintetica e verifica la coerenza normativa e la completezza della bozza.*"

Esempio di domanda

"*Sei un avvocato esperto in diritto civile italiano. In materia di locazioni abitative, con aggiornamento al 2025, considera il caso di un conduttore moroso da tre mesi (gennaio, febbraio, marzo 2025) con contratto registrato a Roma. Puoi fornirmi una bozza sintetica di diffida con riferimenti normativi aggiornati? Verifica inoltre la coerenza e la completezza della bozza*"

🧩 **Schema da ricordare**

R – Ruolo
–
C – Contesto
A – Attori e Fatti
F – Finalità
A – Aggiornamento
R – Richieste specifiche

Test avanzato via *script* Python

Per utilizzare il modello LLaMA 3.1 8B tramite *script* Python, sfruttando l'API locale esposta da Ollama, si può ricorrere a una semplice richiesta HTTP.

Questo è particolarmente utile per chi intende integrare il modello linguistico in flussi di lavoro più automatizzati o in progetti Python di maggiore complessità.

Apriamo il progetto Python creato in precedenza con VS Code, installiamo le librerie necessarie, incolliamo il codice che segue, e avviamo poi lo *script*.

Installazione librerie

Per installare le librerie necessarie, apri Visual Studio Code e segui questi passaggi:

1. Apri la cartella del progetto che contiene il tuo file Python.
2. Apri il terminale interno di VS Code dal menu *Visualizza*, poi *Terminale*.
3. Attiva l'ambiente virtuale prima di procedere con l'installazione (come indicato precedentemente).
4. Copia il comando qui sotto, incollalo nel terminale e premi *Invio*.
5. Attendi che l'installazione venga completata.

pip install requests

Esempio di *script* commentato:

```
import requests   # importa la libreria per effettuare richieste HTTP

import json       # importa la libreria per gestire dati JSON

# Invia una richiesta POST all'API locale di Ollama, specificando il modello e il prompt

response = requests.post("http://localhost:11434/api/generate", json={

   "model": "llama3.1:8b",

   "prompt": "Quali sono i principi fondamentali della Costituzione italiana?"

}, stream=True)  # abilita lo streaming della risposta
```

```
# Legge la risposta riga per riga (stream JSON)

for line in response.iter_lines():

    if line:  # se la riga non è vuota

        data = json.loads(line.decode('utf-8'))  # decodifica e carica il JSON

        if "response" in data:

            print(data["response"], end="", flush=True)  # stampa il testo generato
```

Questo frammento invia un *prompt* giuridico al modello LLaMA 3.1 8B in esecuzione e restituisce la risposta generata riga per riga, in modo corretto per la modalità *streaming* di Ollama. È possibile adattarlo per:

- Elaborare in *batch* un elenco di quesiti
- Integrare il modello in una semplice interfaccia *web*
- Collegarlo a documenti testuali già preprocessati
- Automatizzare una sezione FAQ per Clienti su aree specifiche di diritto

💡 *Assicurati che LLaMA 3.1 8B sia attivo in* background *(Ollama) prima di lanciare lo* script*, per evitare errori di connessione.*

Utilizzi pratici di un LLM locale per studi legali

I Large Language Models (LLM) rappresentano oggi uno strumento innovativo e potente per automatizzare e supportare molte attività tradizionalmente complesse e

dispendiose negli studi legali, migliorando efficienza, qualità e accessibilità dei servizi legali.

1. Generazione di sintesi automatiche di sentenze e atti giudiziari

Gli LLM possono analizzare rapidamente testi lunghi e complessi come sentenze, ordinanze o atti giudiziari, producendo sintesi chiare e comprensibili. Questo consente agli Avvocati di risparmiare tempo prezioso nella fase di studio e preparazione, focalizzandosi sugli aspetti più rilevanti del caso. L'integrazione con sistemi RAG permette di arricchire queste sintesi con riferimenti precisi a documenti e fonti specifiche dello studio, aumentando l'accuratezza e la pertinenza.

2. Spiegazioni didattiche di concetti giuridici complessi

Gli LLM sono in grado di tradurre il linguaggio tecnico-legale in spiegazioni più accessibili, utili sia per la formazione interna dello studio sia per migliorare la comunicazione con i Clienti, spesso non esperti di diritto. Questa funzione favorisce una maggiore comprensione e trasparenza, facilitando l'orientamento del Cliente e la sua partecipazione consapevole al processo legale.

3. Supporto preliminare alla redazione di atti e pareri

Attraverso la generazione automatica di bozze di documenti legali, come pareri, contratti o memorie, gli LLM possono accelerare la fase iniziale della redazione, fornendo una base strutturata da perfezionare e personalizzare. Questo riduce il carico di lavoro ripetitivo e permette agli Avvocati di dedicarsi maggiormente all'analisi critica e alla strategia legale.

4. Verifica automatica di incongruenze e anomalie nei documenti

Gli strumenti basati su LLM possono essere utilizzati per il controllo qualità dei documenti legali, identificando incongruenze testuali, clausole mancanti o errori

formali. Questa funzione è particolarmente utile per studi di dimensioni medio-piccole che non dispongono di *team* dedicati esclusivamente al controllo documentale, migliorando l'affidabilità e la sicurezza delle pratiche.

5. Accesso semplificato e democratizzazione del diritto

L'adozione degli LLM favorisce anche la creazione di *chatbot* e assistenti virtuali che offrono un primo orientamento giuridico generale ai cittadini, facilitando l'accesso al diritto e ai servizi legali, soprattutto per chi non ha competenze specifiche o risorse economiche per consulenze immediate, senza ovviamente sostituire il professionista.

6. Ottimizzazione della ricerca giuridica e della gestione documentale

Integrati con sistemi RAG e tecniche di *prompt engineering*, gli LLM potenziano la ricerca giuridica interna, permettendo di interrogare grandi archivi documentali e normative in modo conversazionale e contestualizzato. Ciò consente di individuare rapidamente precedenti, norme rilevanti e dottrina applicabile, migliorando la qualità delle analisi e delle strategie legali.

Impatto anche per studi di piccole dimensioni

Anche studi legali di dimensioni ridotte possono beneficiare di queste tecnologie, grazie alla disponibilità crescente di soluzioni *open source* e strumenti accessibili che non richiedono grandi investimenti infrastrutturali o *team* IT dedicati.

L'uso mirato e contestualizzato degli LLM consente di aumentare la produttività, ridurre i tempi di lavoro e migliorare la qualità del servizio offerto ai Clienti, livellando il campo competitivo con studi più grandi.

Considerazioni finali

L’integrazione di un LLM locale negli studi legali apre la strada a una trasformazione profonda della professione, combinando automazione, supporto decisionale e maggiore accessibilità.

Tuttavia, è fondamentale accompagnare questa innovazione con una gestione attenta dei rischi, assicurando in particolare la tutela della privacy, la qualità dei dati e la validazione professionale delle risposte generate, per garantire un uso responsabile e conforme alle normative vigenti.

Limiti di LLaMA 3.1 8B (e di altri modelli “generalisti”) nell’ambito legale italiano

1. Conoscenza giuridica limitata e generalista

LLaMA 3.1 8B è un modello linguistico di grandi dimensioni addestrato su un vasto *corpus* multilingue e generalista, che include testi in varie lingue e ambiti. Tuttavia, non è stato addestrato specificamente su fonti giuridiche italiane aggiornate, né su banche dati normative o giurisprudenziali italiane.

Di conseguenza, le risposte fornite dal modello possono risultare superficiali, incomplete o imprecise quando si tratta di diritto italiano, soprattutto per norme, prassi e interpretazioni più recenti o specifiche del nostro ordinamento.

2. Mancanza di aggiornamento normativo e giurisprudenziale in tempo reale

Il modello si basa su dati di *training* con una data di taglio (*cut-off*) e non può accedere a informazioni aggiornate in tempo reale, né a banche dati giuridiche o a Internet. Questo è un limite critico in ambito legale, dove le normative e la giurisprudenza evolvono costantemente.

Pertanto, le risposte possono non riflettere le ultime modifiche legislative o gli orientamenti giurisprudenziali più recenti.

3. Complessità e specificità del linguaggio giuridico italiano

Il diritto italiano utilizza un linguaggio tecnico, spesso complesso e con sfumature interpretative che richiedono una profonda conoscenza giuridica e contestuale.

LLaMA 3.1, pur essendo multilingue e capace di gestire l'italiano, non sempre coglie appieno queste sfumature, rischiando di fornire risposte generiche o non pienamente aderenti al contesto normativo italiano.

4. Necessità di validazione professionale

Data la natura generalista e i limiti sopra indicati, le risposte generate da LLaMA 3.1 8B devono sempre essere verificate e interpretate da un Avvocato o esperto legale prima di essere utilizzate in ambito professionale o decisionale.

Il modello non può sostituire l'analisi giuridica critica, l'interpretazione normativa e la valutazione del contesto specifico che solo un professionista può fornire.

5. Assenza di personalizzazione e integrazione con documenti specifici

Senza un sistema di tipo RAG (Retrieval-Augmented Generation) che integri il modello con i documenti, le sentenze e le normative specifiche dello studio legale, LLaMA 3.1 8B non può "conoscere" o utilizzare direttamente il patrimonio documentale specifico.

Questo limita la sua efficacia in applicazioni pratiche come la redazione assistita di pareri o la verifica di documenti, dove la conoscenza contestualizzata è fondamentale.

In sintesi

LLaMA 3.1 8B rappresenta uno strumento potente e versatile, ma nel contesto del diritto italiano presenta limiti significativi legati alla sua natura generalista e alla mancanza di dati specifici e aggiornati.

Per un uso efficace in ambito legale è indispensabile integrare il modello con sistemi di recupero e contestualizzazione dei documenti (RAG) e mantenere sempre un controllo professionale sulle risposte generate.

LLM locale e studi legali: conclusione

L'esecuzione di un LLM locale con Ollama costituisce un importante passo nella costruzione di un'infrastruttura legale potenziata dall'IA. Questo tipo di approccio consente di:

- Mantenere il pieno controllo sui dati e la riservatezza dei documenti
- Sperimentare l'uso dell'intelligenza artificiale senza dover ricorrere a soluzioni *cloud* esterne
- Prepararsi all'integrazione di sistemi RAG più evoluti e automatizzati

Con strumenti semplici e *open source*, anche uno studio legale senza team IT può oggi testare modelli linguistici evoluti in modo locale, protetto e indipendente.

Questo approccio apre la strada a numerosi sviluppi futuri, come l'indicizzazione intelligente dei documenti, l'interrogazione conversazionale dei fascicoli o la generazione automatica di analisi e relazioni.

Nel prossimo articolo vedremo come preparare e pulire i documenti PDF in modo da renderli compatibili con un sistema RAG, sfruttando OCR e tecniche NLP.

8. Parsing dei PDF legali con OCR e pulizia del testo

Che cos'è il parsing?

Nel contesto dell'elaborazione dei documenti legali, il termine *parsing* indica l'intero processo di analisi e trasformazione di un documento testuale in una forma strutturata, adatta a essere elaborata da un modello linguistico o un motore RAG. Le fasi principali del parsing sono:

1. **Estrazione del contenuto**: recupero del testo dal documento, sia esso digitale (PDF nativo) o immagine (via OCR).
2. **Pulizia del testo**: rimozione di elementi non rilevanti (es. intestazioni, watermark, firme, numeri pagina).
3. **Segmentazione**: suddivisione del testo in blocchi logici (*chunk*), utili per l'analisi semantica e l'indicizzazione.

Una buona *pipeline* di parsing è la base per costruire sistemi intelligenti in grado di comprendere e sfruttare documenti legali complessi.

Introduzione

Nota terminologica

OCR (Optical Character Recognition): tecnologia che consente di convertire immagini contenenti testo (es. PDF scannerizzati) in testo digitale modificabile.

NLP (Natural Language Processing): insieme di tecniche e strumenti che permettono ai computer di comprendere, analizzare e generare il linguaggio umano in modo utile, ad esempio per classificare, riassumere o cercare contenuti giuridici.

Nell'ambito dell'adozione di sistemi RAG (Retrieval-Augmented Generation) locali negli studi legali, una delle fasi più critiche riguarda la **preparazione dei dati**, in particolare l'estrazione del testo da documenti PDF.

I documenti legali possono essere sia in formato digitale nativo (PDF generati da Word o da software gestionali), sia scannerizzati come immagini.

La capacità di processare PDF in modo accurato rappresenta dunque un passaggio essenziale per alimentare un sistema basato su LLM.

Questo ottavo post ci guida passo dopo passo nel parsing dei PDF legali, distinguendo tra documenti nativi e scannerizzati.

Verranno illustrati strumenti come Tesseract per l'OCR, PyMuPDF e pdfminer per l'estrazione del testo, oltre a tecniche di pulizia del testo.

Esempi di parsing

Estrazione di testo da PDF nativi

Spiegazione tecnica

In questo esempio di parsing vengono mostrati due strumenti per l'estrazione del testo da PDF digitali nativi, cioè file generati digitalmente (non scannerizzati).

- PyMuPDF (fitz): consente di aprire un PDF, leggere ogni pagina e ricavare il testo in modo rapido. La funzione `get_text()` estrae tutto il contenuto testuale della pagina.

- `pdfminer.six`: è una libreria più sofisticata per casi in cui il *layout* del documento è complesso (colonne, tabelle, strutture giuridiche complesse). La funzione `extract_text()` estrae tutto il testo del PDF in una sola chiamata.

Entrambi gli strumenti funzionano in ambiente Windows, sono *open source* e integrabili in flussi Python automatizzati per alimentare un sistema RAG.

Guida pratica: installazione librerie, creazione ed esecuzione script python

Installazione delle librerie con pip

Come già illustrato nei post 6 e 7:

1. Apri Visual Studio Code.
2. Apri la cartella del progetto che contiene il file python e l'ambiente virtuale.
3. Apri il terminale integrato (dal menu Visualizza > Terminale).
4. Attiva l'ambiente virtuale.
5. Installa le librerie necessarie eseguendo il comando:
   `pip install pymupdf pdfminer.six`

Creazione e salvataggio dello script Python

1. All'interno della cartella del progetto, crea un nuovo file chiamato `estrai_testo.py`.
2. Copia e incolla il codice python di esempio che trovi di seguito nel file e salvalo.

Esecuzione dello script Python

Una volta creato il file .py con l'esempio, nel terminale di VS Code, esegui lo script con:

python estrai_testo.py

Questo comando:

- eseguirà lo script,
- aprirà il file `contratto.pdf` (che deve trovarsi *nella stessa cartella* del file estrai_testo.py che hai creato),
- estrarrà il testo dal PDF, lo stamperà nel terminale e lo salverà anche in un file di testo.

Esempio di parsing con PyMuPDF (script python):

```
import fitz  # PyMuPDF

# Apri il file PDF da cui vuoi estrarre il testo

doc = fitz.open("contratto.pdf")

# Scorri tutte le pagine del documento

with open("testo_estratto.txt", "w", encoding="utf-8") as f:

 for page in doc:

    # Estrai il testo dalla pagina corrente

    text = page.get_text()

    # Stampa il contenuto testuale della pagina

    print(text)
```

```
    # Crea un file con il testo estratto
    f.write(text)
# Chiudi il documento per liberare risorse
doc.close()
```

Cosa fa questo codice:

- Apre il file `contratto.pdf`.
- Estrae il testo da ogni pagina usando `get_text()`.
- Stampa il contenuto sul terminale e lo salva in un file .txt.
- Infine, chiude il file per buona prassi.

Esempio di parsing con pdfminer:

```
from pdfminer.high_level import extract_text
text = extract_text("contratto.pdf")
print(text)
with open("testo_estratto.txt", "w", encoding="utf-8") as f:
    f.write(text)
```

OCR su PDF scannerizzati con Tesseract

Perché è necessario

Molti atti giudiziari, contratti firmati o documenti arrivano in formato immagine. Senza OCR, questi documenti restano inaccessibili agli LLM e inutilizzabili nei processi NLP.

Per l'installazione delle librerie necessarie e la creazione del file .py dello script python seguire il medesimo procedimento descritto sopra.

Installazione di Tesseract OCR su Windows con lingua italiana

Per utilizzare Tesseract OCR in italiano su Windows, segui questi passaggi:

1. Scarica l'installer di Tesseract

Vai alla pagina ufficiale dei rilasci di Tesseract per Windows su GitHub:

https://github.com/UB-Mannheim/tesseract/wiki

Scarica la versione più recente dell'installer (tesseract-ocr-w64-setup.exe) e avviala. Durante l'installazione, puoi scegliere le lingue aggiuntive da installare: assicurati di selezionare anche "Italian".

Installazione delle librerie necessarie

pip install pymupdf pytesseract Pillow

Esempio pratico con PyMuPDF e Tesseract (OCR)

Questo esempio consente di eseguire l'OCR direttamente su ogni pagina di un PDF scannerizzato, utilizzando PyMuPDF per convertire le pagine in immagini e Tesseract per estrarre il testo.

```
import fitz  # PyMuPDF

import pytesseract

from PIL import Image

from io import BytesIO

# Percorso all'eseguibile di Tesseract (modificare se necessario)

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

# Apri il file PDF scannerizzato

pdf_document = fitz.open('documento_scannerizzato.pdf')

# Scorri ogni pagina del PDF

with open("testo_estratto.txt", "w", encoding="utf-8") as f:

  for i, page in enumerate(pdf_document):

    # Crea una rappresentazione raster della pagina (300 DPI per buona qualita' OCR)
```

```
    pix = page.get_pixmap(dpi=300)

    # Converte l'immagine in oggetto compatibile con PIL

    img = Image.open(BytesIO(pix.tobytes("png")))

    # Applica OCR con lingua italiana

    text = pytesseract.image_to_string(img, lang='ita')

    # Stampa il testo estratto

    print(f"--- Pagina {i+1} ---")

    print(text)

    # Salva su file il testo estratto

    f.write(text)

pdf_document.close()
```

Cosa fa questo codice

- Apre il PDF con PyMuPDF (nell'esempio il file documento_scannerizzato.pdf che si trova nella stessa cartella del file .py)
- Converte ogni pagina in immagine ad alta risoluzione
- Usa Tesseract per eseguire OCR sull'immagine
- Stampa il testo riconosciuto da ogni pagina e lo salva in un file .txt nella medesima cartella.

Parsing dei PDF legali: pulizia del testo

Che cos'è la pulizia del testo?

La pulizia del testo (*text cleaning*) è il processo con cui si rimuovono elementi indesiderati da un testo grezzo, come intestazioni ripetute, firme, numeri di pagina, *watermark* o errori derivanti da OCR. L'obiettivo è ottenere un testo coerente, leggibile e pronto per l'analisi automatica.

In ambito legale, la pulizia è fondamentale per evitare che rumore testuale interferisca con la comprensione semantica da parte di modelli NLP o motori RAG.

Un testo pulito migliora l'accuratezza delle risposte generate, la pertinenza dei documenti recuperati e l'efficacia delle sintesi automatiche.

Obiettivi principali della pulizia

- Eliminare intestazioni, numeri di pagina, *watermark*
- Uniformare gli spazi e correggere gli errori OCR
- Conservare riferimenti numerici rilevanti (es. articoli di legge, commi, date)

Funzione di pulizia avanzata

```
# -*- coding: utf-8 -*-

import re
```

```
import unicodedata


def pulisci_testo_legale(testo):
    # Normalizza caratteri Unicode (es. accenti strani da OCR)
    testo = unicodedata.normalize("NFKC", testo)
    # Rimuove intestazioni tipo "Studio Legale Rossi Associati"
    testo = re.sub(r'Studio\s+Legale[\w\s&\.]*', '', testo, flags=re.IGNORECASE)
    # Rimuove numeri di pagina "Pagina X di Y"
    testo = re.sub(r'(?i)pagina\s+\d+\s+di\s+\d+', '', testo)
    # Rimuove etichette come [BOZZA], [NON UFFICIALE], [RISERVATA]
    testo = re.sub(r'\[?\b(?:BOZZA|NON\s+UFFICIALE|RISERVATA)\b\]?', '', testo,
flags=re.IGNORECASE)
    # Rimuove firme tipo "Avv. Mario Rossi" (anche tutto maiuscolo)
    testo = re.sub(r'Avv\.\s+[A-Za-zÀ-ÖØ-öø-ÿ]+(?:\s+[A-Za-zÀ-ÖØ-öø-ÿ]+)+', '', testo,
flags=re.IGNORECASE)
    # Riduce spazi multipli a uno solo
    testo = re.sub(r'[ \t]{2,}', ' ', testo)
    # Riduce righe vuote multiple a una sola
```

```
    testo = re.sub(r'\n\s*\n+', '\n\n', testo)

    # Rimuove spazi all'inizio e alla fine

    return testo.strip()

# ESEMPIO DI UTILIZZO

testo_originale = """

Studio Legale Rossi Associati

PAGINA 1 DI 5

Copia riservata al Cliente

[BOZZA NON UFFICIALE]

Il contratto viene definito dall'art. 1321 c.c.

come l'accordo di  due  o  più  parti  per  costituire,

regolare o estinguere tra loro un rapporto giuridico patrimoniale...

Data: 4 maggio 2025

Avv. Maria Bianchi

"""

testo_pulito = pulisci_testo_legale(testo_originale)

print("TESTO ORIGINALE:")
```

```
print(testo_originale)


print("\nTESTO PULITO:")

print(testo_pulito)
```

Conclusione

Il parsing accurato dei PDF legali è un passaggio fondamentale per costruire un sistema RAG realmente efficace.

Saper distinguere tra documenti nativi e scannerizzati, utilizzare strumenti adeguati e pulire correttamente i testi consente di ottenere basi dati affidabili per la ricerca giuridica, la sintesi e l'elaborazione automatica.

Nel prossimo post vedremo come costruire un semplice motore RAG locale di esempio collegando questi testi a un LLM tramite FAISS e LangChain.

9. Implementazione locale di un motore RAG base per studi legali

Questo nono post della guida pratica accompagna gli studi legali nell'adozione concreta dell'Intelligenza Artificiale generativa.

In particolare, ci concentriamo sull'implementazione in ambiente Windows di un **semplice sistema RAG (Retrieval-Augmented Generation) locale** di esempio, utilizzando LangChain, FAISS e Ollama.

🎯 Obiettivo del post

Fornire una guida chiara e commentata per costruire un semplice motore RAG locale d'esempio in grado di interrogare **archivi interni di documenti legali** e **generare risposte pertinenti**, con attenzione a riservatezza, semplicità e riutilizzabilità del codice.

🧩 Concetti chiave

- **RAG**: sistema che unisce il recupero semantico da un archivio indicizzato con la generazione di testo tramite LLM.
- **LangChain**: framework per creare *pipeline* tra LLM, archivi e altri strumenti NLP.
- **FAISS**: libreria per l'indicizzazione e la ricerca veloce di vettori.
- **Ollama**: ambiente locale per eseguire LLM su dispositivi propri.

💼 Prerequisiti

Per poter utilizzare correttamente lo *script* che crea un motore RAG base, è necessario predisporre l'ambiente nel modo seguente (vedi post 6):

- Un computer con **Windows 10 o superiore**.
- **Python 3.10 o versione compatibile** installato.
- **Visual Studio Code** configurato con ambiente virtuale attivo.
- Il software **Ollama** installato e funzionante.
- I seguenti modelli scaricati con Ollama tramite terminale:

ollama pull llama3.1:8b

ollama pull nomic-embed-text

- Una cartella locale che contenga i documenti PDF legali da interrogare. Questa cartella può essere:
    - denominata `documenti_legali` e posizionata **nella stessa cartella del file python (.py)** dello script;
    - oppure configurata usando un **percorso assoluto in formato Windows**, preferibilmente come **raw string**, ad esempio:

directory = r"C:\Users\NomeUtente\Documents\StudioLegale\documenti_legali"

🔍 **Nota per utenti non esperti**: per sapere qual è il percorso esatto della cartella, fai clic con il tasto destro su di essa in Esplora File e seleziona "Copia come percorso". Ricordati di anteporre `r` alla stringa incollata nello script, oppure di raddoppiare ogni barra `\` per evitare errori.

🛠️ Installazione dei pacchetti necessari

Se stai utilizzando Visual Studio Code e non hai familiarità con il terminale, segui questi semplici passaggi:

1. Apri Visual Studio Code e assicurati di avere attivato l'ambiente virtuale come spiegato nel **post 6**.
2. Clicca sul menu **Visualizza > Terminale**.
3. Nella parte inferiore dello schermo si aprirà il terminale.
4. Copia e incolla il comando seguente nel terminale e premi `Invio`:

pip install langchain langchain-community faiss-cpu langchain-ollama PyMuPDF langchain-text-splitters

Queste librerie servono per:

- orchestrare il sistema (LangChain);
- creare rappresentazioni numeriche dei testi (*embedding*);
- effettuare ricerche semantiche (FAISS);
- far funzionare i modelli in locale (Ollama);
- leggere i contenuti dei PDF (PyMuPDF).

Una volta completata l'installazione, puoi passare direttamente allo script di esempio.

Script completo con spiegazioni per il motore RAG base

Esecuzione di esempio

Immagina di avere nella cartella `./documenti_legali` un file PDF contenente il testo della Costituzione italiana. Se esegui lo script e scrivi nel terminale la domanda:

Quali sono i principi fondamentali della Costituzione italiana?

il sistema estrae il testo dal documento PDF, lo analizza semanticamente, individua i blocchi rilevanti e, tramite LLM, fornisce una risposta come questa:

Secondo l'articolo 1 della Costituzione italiana, i principi fondamentali sono:

** L'Italia è una Repubblica democratica, fondata sul lavoro.*

** La sovranità appartiene al popolo, che la esercita nelle forme e nei limiti della Costituzione.*

Inoltre, l'articolo 2 precisa che la Repubblica riconosce e garantisce i diritti inviolabili dell'uomo, sia come singolo sia nelle formazioni sociali ove si svolge la sua personalità, e richiede l'adempimento dei doveri inderogabili di solidarietà politica, economica e sociale.

Infine, l'articolo 3 afferma che tutti i cittadini hanno pari dignità sociale e sono eguali davanti alla legge, senza discriminazioni di sorta.

Confrontiamo questa risposta con quella data alla medesima domanda dallo stesso LLM "generalista" prima di "conoscere" il file contenente la Costituzione (post 7).

Ora il modello fornisce una risposta più precisa e pertinente.

Lo script

Lo script seguente costruisce un motore RAG locale, passo dopo passo. I documenti PDF vengono letti, segmentati in blocchi, trasformati in vettori semantici, indicizzati e interrogati tramite un LLM locale.

Lo script lavora con *PDF nativi*. Per l'estrazione del testo da file immagine o PDF scannerizzati, e per la pulizia preliminare del testo, si rimanda al post 8.

```
# -- a) Importazioni --

import os

import fitz  # PyMuPDF per leggere PDF

from langchain_core.documents import Document

from langchain_ollama import OllamaEmbeddings

from langchain_text_splitters import RecursiveCharacterTextSplitter

from langchain_community.vectorstores import FAISS

from langchain_core.prompts import ChatPromptTemplate

from langchain_ollama.llms import OllamaLLM


# -- b) Caricamento PDF --

directory = "./documenti_legali"

documents = []

for filename in os.listdir(directory):

    if filename.endswith(".pdf"):
```

```
        filepath = os.path.join(directory, filename)
        with fitz.open(filepath) as pdf:
            text = "\n".join(page.get_text() for page in pdf)
            documents.append(Document(page_content=text, metadata={"source": filename}))

# -- c) Segmentazione in chunk --
splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=100)
split_docs = splitter.split_documents(documents)

# -- d) Embedding semantico --
embeddings = OllamaEmbeddings(model="nomic-embed-text")

# -- e) Vector store con FAISS --
vector_store = FAISS.from_documents(split_docs, embeddings)

# -- f) Query e recupero documenti simili --
query = input("Ciao, sono il tuo assistente legale. Inserisci la tua domanda: ")
```

```
docs = vector_store.similarity_search(query)


# Estrai il contenuto testuale dai documenti

context = "\n\n".join([f"FONTE: {doc.metadata['source']}\nCONTENUTO:
{doc.page_content}" for doc in docs])


# -- g) Generazione risposta con LLM --

prompt = ChatPromptTemplate.from_template ("""

Sei un assistente giuridico specializzato in diritto italiano.

Usa esclusivamente il testo fornito come contesto per elaborare le tue risposte.


CONTESTO:

{context}


DOMANDA: {question}

RISPOSTA:

""")
```

```
model = OllamaLLM(model="llama3.1:8b")

chain = prompt | model

response = chain.invoke({"question": query, "context": context})

print(response)
```

Glossario essenziale:

- Chunk: un frammento di testo (es. 1000 caratteri) in cui viene suddiviso il documento per facilitare l'elaborazione da parte dell'IA. Serve per non superare i limiti del modello e per mantenere il contesto logico.
- Embedding: la trasformazione di un testo in un vettore numerico che ne rappresenta il significato. Più due testi sono simili, più i loro vettori saranno vicini nello spazio matematico.
- Vector store: un archivio indicizzato di vettori (testi trasformati in numeri) che permette di cercare i documenti più simili a una domanda.
- LLM: acronimo di "Large Language Model". È il cervello generativo che, dato un contesto (come i chunk trovati), formula una risposta in linguaggio naturale.
- LangChain: un framework Python che consente di mettere insieme tutti questi strumenti in modo fluido e strutturato.

Questo script è composto da sette fasi principali:

1. **Importazioni**: in questa fase vengono caricate le librerie necessarie. Queste librerie permettono di:
    1. leggere i PDF;
    2. dividere il testo in parti più gestibili (*chunk*);
    3. trasformare i testi in vettori numerici comprensibili dal sistema;

4. costruire una base dati semantica interrogabile;
5. generare una risposta coerente tramite un modello di linguaggio.

2. **Caricamento PDF**: lo script accede a una cartella locale contenente i PDF e li apre uno a uno. Utilizza PyMuPDF per leggere il testo da ogni pagina. Ogni file viene trasformato in un oggetto strutturato che conserva sia il contenuto testuale che informazioni utili (es. il nome del file).
3. **Segmentazione** in ***chunk***: i testi estratti vengono suddivisi in blocchi da 1000 caratteri con 100 caratteri di sovrapposizione tra un blocco e l'altro. Questo consente di preservare la continuità delle informazioni e ottenere risposte più precise. È come dividere un lungo paragrafo in sezioni leggibili senza perdere il filo logico.
4. ***Embedding* semantico**: ogni blocco di testo viene convertito in un vettore numerico grazie al modello `nomic-embed-text`. Questo passaggio è essenziale per consentire al sistema di confrontare testi in modo "intelligente", basandosi sul significato e non solo sulle parole.
5. Creazione del ***vector store***: i vettori ottenuti vengono inseriti in un archivio indicizzato chiamato *vector store*, grazie alla libreria FAISS. Questo archivio consente di effettuare ricerche veloci e accurate, trovando i blocchi di testo più simili a una domanda posta in linguaggio naturale.
6. **Ricerca**: quando l'utente formula una domanda, il sistema cerca nel vector store i blocchi di testo più pertinenti. Non si tratta di una ricerca per parole chiave, ma di una ricerca semantica: il sistema capisce il significato della domanda e recupera ciò che realmente serve.
7. **Generazione della risposta**: i risultati recuperati vengono passati a un LLM locale (llama3.1:8b), che genera una risposta strutturata e in linguaggio naturale alla domanda posta. In questa fase, i documenti trovati non vengono semplicemente copiati, ma vengono usati come "contesto" per istruire il modello. Il modello prende la domanda dell'utente e costruisce una risposta sfruttando le informazioni recuperate. Questo è il cuore del sistema RAG: il recupero (***retrieval***)

trova ciò che serve, e la generazione (***generation***) riformula in modo chiaro e coerente. È come se un assistente legale umano, leggendo i documenti più rilevanti, producesse una sintesi precisa su misura per la domanda ricevuta. L'intelligenza artificiale, grazie a questo meccanismo, diventa realmente utile e affidabile, offrendo risposte basate sui contenuti reali dell'archivio legale interno dello studio.

Questa sequenza consente di costruire un semplice assistente giuridico che consulta archivi interni e fornisce risposte basate su fonti testuali reali.

Si tratta ovviamente di un semplice esempio, un punto di partenza.

🔧 Come usare lo script per il motore RAG base in Visual Studio Code

Questo script può essere copiato in un file `motore_rag.py` ed eseguito in Visual Studio Code (VS Code).

Nel **post 6** abbiamo già spiegato:

- come installare VS Code,
- come creare un progetto python,
- e come attivare un ambiente virtuale.

Una volta completati quei passaggi:

1. Crea un nuovo file `motore_rag.py`
2. Incolla il codice dello script e salva il file
3. Installa i pacchetti nel terminale virtuale
4. Esegui lo script con il comando `python motore_rag.py`

⚖️ Vantaggi e limiti di un sistema RAG locale per lo studio legale

Un sistema RAG locale costruito su misura per uno studio legale italiano presenta numerosi vantaggi, ma va considerato nel contesto di altre soluzioni disponibili, come quelle **ibride (parzialmente su cloud)** o **verticali (pensate specificamente per il settore legale)**.

✅ Vantaggi principali

- **Privacy garantita**: tutti i documenti, le domande e le risposte restano all'interno della macchina locale dello studio. Questo è fondamentale per rispettare il segreto professionale, il GDPR e l'etica forense. Nessun dato viene inviato a server esterni.
- **Velocità di consultazione**: il sistema lavora su archivi interni indicizzati semanticamente, consentendo interrogazioni rapide anche su grandi quantità di testi. Le domande possono essere formulate in linguaggio naturale e ottenere risposte centrate.
- **Flessibilità e controllo**: il sistema è completamente personalizzabile. Si può adattare a vari ambiti del diritto (es. immigrazione, civile, penale, lavoro) semplicemente caricando i documenti relativi. Inoltre, può essere esteso con nuove funzionalità, prompt specifici, o interfacce utente.
- **Indipendenza tecnologica**: l'uso di strumenti open-source e l'esecuzione locale svincolano lo studio da licenze costose o soluzioni chiuse. Lo strumento resta pienamente controllabile e modificabile.

⚠️ Limiti da considerare

- **Maggiori responsabilità di gestione**: serve un minimo di competenza tecnica per mantenere aggiornato il sistema e garantire il buon funzionamento (es. aggiornamento dei modelli, indicizzazione documenti).
- **Assenza di ottimizzazioni legali verticali**: rispetto a piattaforme commerciali verticali, un sistema generico RAG locale non include per default modelli addestrati sul diritto italiano o su giurisprudenza specifica (ma si può arrivare a quel livello con estensioni mirate).
- **Accessibilità limitata**: un sistema locale funziona solo nel contesto dove è stato installato. Le soluzioni ibride offrono invece l'accesso multi-dispositivo o da remoto.

In sintesi, un RAG locale è una scelta ottima per studi legali che vogliono autonomia, privacy e personalizzazione, ma va valutato attentamente rispetto a bisogni specifici, risorse disponibili e alternative sul mercato.

🚀 Come migliorare e potenziare l'assistente legale AI di base

Una volta compreso e testato il sistema RAG d'esempio presentato in questo post, è possibile ampliarlo gradualmente per ottenere un assistente legale ancora più completo, efficiente e utile nello studio. Ecco i principali miglioramenti che consigliamo, spiegati in modo semplice:

1. **Espandere la base documentale**: puoi aggiungere nuove categorie di documenti (atti, pareri, giurisprudenza, normativa), anche suddividendole in sottocartelle tematiche. Il motore RAG potrà essere istruito a distinguere tra le fonti e a usare quella più adatta alla domanda.
2. **Interfaccia utente**: puoi costruire una semplice interfaccia grafica (GUI) con Streamlit o Gradio, così da non dover usare il terminale.

3. **Controllo della pertinenza**: puoi configurare il sistema per restituire, oltre alla risposta, anche i documenti effettivamente consultati, così da garantire la massima trasparenza e verificabilità delle risposte.
4. **Aggiungere sempre un livello di verifica umana**: prima che una risposta venga utilizzata per finalità professionali occorre proporla sempre a un avvocato dello studio per la **validazione**. Questo garantisce rigore e tutela da errori.
5. **Prompt personalizzati**: è possibile costruire prompt su misura per ogni tipo di richiesta: redazione di pareri, confronto normativo, sintesi di sentenze, ecc. Questo rende il sistema ancora più flessibile.
6. **Salvataggio e riutilizzo delle sessioni**: con opportune modifiche, si può tenere traccia delle domande e delle risposte fornite, così da creare un archivio consultabile delle ricerche effettuate in studio.

Questi miglioramenti possono essere introdotti anche uno per volta, a seconda delle competenze disponibili e del tempo che si vuole investire. L'importante è partire da una base solida e locale come quella presentata qui: ogni estensione diventerà poi un naturale passo successivo.

🧠 Come migliorare la qualità delle risposte generate dal motore RAG base

Anche partendo da uno script di base, è possibile ottenere risposte più pertinenti, affidabili e formalmente corrette. Ecco alcune strategie semplici ma efficaci per migliorare la qualità delle risposte fornite dal sistema:

1. **Curare la qualità del testo nei PDF**: il sistema RAG può restituire buone risposte solo se i documenti da cui apprende sono scritti chiaramente. Evita PDF con testo malformattato o OCR impreciso. È utile usare strumenti come Tesseract (vedi post 8) per pulire e standardizzare i contenuti.

2. **Ottimizzare i prompt**: istruzioni chiare nel prompt (es. “Usa un linguaggio giuridico formale”, oppure “Cita gli articoli pertinenti”) orientano il modello verso uno stile più preciso e contestuale. Il prompt attuale può essere ulteriormente raffinato in base al tipo di richiesta (pareri, analisi normativa, sintesi giurisprudenziale). Si affronterà in seguito nel dettaglio il tema del *legal prompting*.
3. **Limitare il contesto**: invece di passare tutti i chunk trovati, si possono limitare a quelli con maggiore similarità (es. top 3). Questo aiuta il modello a concentrarsi sulle informazioni più rilevanti, evitando risposte vaghe.
4. **Impostare il modello con un tono specifico**: tramite parametri del modello (come `system` in `OllamaLLM`) si può chiedere di mantenere sempre un tono formale, con terminologia legale e senza generalizzazioni.
5. **Controllare i limiti di token**: se il contesto è troppo lungo, può essere tagliato. È importante monitorare quanto testo si fornisce al modello per rimanere entro i limiti e garantire risposte complete.
6. **Testare più modelli**: LLaMA è un buon modello generale, ma si può valutare l’uso di LLM giuridici addestrati su *corpora* legali (es. modelli basati su sentenze, diritto italiano o europeo).

Applicando anche solo alcune di queste strategie, la qualità delle risposte migliora sensibilmente, rendendo il sistema più utile e affidabile nello studio.

📊 Come migliorare l’indicizzazione con FAISS e ottenere recuperi più efficaci

Per rendere il sistema ancora più potente ed efficiente, è possibile ottimizzare l’indicizzazione dei documenti e le ricerche nel vector store FAISS. Ecco alcune strategie, spiegate in modo semplice:

1. **Pre-elaborazione del testo**: prima di generare i chunk e gli embeddings, è utile rimuovere elementi superflui come caratteri speciali, numerazioni inutili o note a piè di pagina (si veda il post 8). Standardizzare punteggiatura e maiuscole/minuscole migliora la consistenza e quindi anche la qualità dell'indice vettoriale.
2. **Aggiunta di metadati rilevanti**: ogni documento indicizzato può includere metadati (come fonte normativa, anno, ambito tematico), che aiutano a filtrare e contestualizzare meglio i risultati. Ad esempio, potresti recuperare solo i documenti di diritto civile o solo le versioni più aggiornate.
3. **Salvataggio e aggiornamento dell'indice FAISS**: salvare l'indice su disco consente di riutilizzarlo senza doverlo ricostruire ogni volta. È anche possibile aggiornarlo periodicamente con nuovi documenti, mantenendo il sistema sempre aggiornato.
4. **Regolare i parametri di ricerca**: limitare il numero di risultati restituiti (ad esempio i 3 più simili) aiuta il modello generativo a concentrarsi sul contesto davvero utile, migliorando la precisione della risposta.
5. **Sovrapposizione tra chunk ben calibrata**: una buona sovrapposizione evita che concetti importanti vengano tagliati tra due segmenti. Può essere utile aumentarla per testi normativi complessi e ridurla per testi più narrativi.
6. **Ricerca ibrida (semantica + keyword)**: per una copertura più completa, si può combinare la ricerca semantica con una ricerca per parole chiave. Questo approccio è particolarmente utile in ambito giuridico, dove il significato e il lessico specifico hanno entrambi valore. LangChain supporta la combinazione di più meccanismi di retrieval in modo coordinato.

Con questi accorgimenti, il tuo vector store diventa più efficiente, le ricerche più intelligenti e il sistema complessivamente più reattivo e affidabile.

📌 Cosa puoi fare ora con il motore RAG base

✅ Puoi caricare copia di documenti di prova nella cartella `./documenti_legali`

✅ Esegui lo script di base in VS Code con una domanda reale

✅ Verifica la qualità della risposta

✅ Applica uno o più miglioramenti suggeriti nei paragrafi precedenti (es. prompt personalizzati, aggiunta metadati, ricerca ibrida)

✅ Considera di salvare l'indice FAISS per riutilizzarlo e migliorare la velocità

✅ Conclusione

Il sistema RAG locale presentato in questo post rappresenta un semplice e mero esempio, un punto di partenza per comprendere l'IA applicata al settore legale e sperimentare l'implementazione di un assistente legale basato sull'intelligenza artificiale.

È semplice, estendibile, e rispettoso delle esigenze di privacy e controllo dei dati.

Ogni parte del codice è documentata per consentire agli Avvocati, anche senza esperienza tecnica, di comprendere e personalizzare lo strumento.

Non è ovviamente idoneo a un uso professionale.

Nel prossimo post vedremo come migliorare le performance del RAG costruendo prompt intelligenti e su misura per lo studio legale.

10. Legal prompting

Introduzione al legal prompting

Glossario essenziale

- **Prompt**: istruzione in linguaggio naturale che guida l'IA nella generazione del testo.
- **LLM (Large Language Model)**: modello linguistico di grandi dimensioni, in grado di comprendere e generare testi complessi.
- **Token**: unità testuale (parola o parte di parola); ogni modello ha un limite massimo di token gestibili.
- **LangChain**: libreria Python per costruire flussi AI strutturati e integrabili.
- **FAISS**: libreria per la ricerca vettoriale veloce, utile nei sistemi RAG.
- **RAG (Retrieval-Augmented Generation)**: tecnica che combina recupero documentale e generazione AI, aumentando accuratezza e contesto.

Mappa del flusso: come funziona il legal prompting in uno studio legale che usa un sistema RAG

Documento legale (PDF, sentenza, atto)

↓

Pre-processing (OCR, parsing, pulizia testo)

↓

Prompt costruito su misura (es. sintesi, parere, confronto)

↓

Modello linguistico locale (es. Mistral via Ollama)

↓

Output testuale (riassunto, schema, risposta giuridica)

↓

Utilizzo nel flusso di lavoro dello studio (documenti, email, atti...)

sotto la supervisione umana

Questa catena di passaggi è alla base dell'integrazione tra intelligenza artificiale e attività giuridica: ogni anello dipende dalla qualità del precedente, in particolare dalla progettazione del prompt.

In questo articolo ci occupiamo dunque di un aspetto fondamentale nell'adozione dell'intelligenza artificiale generativa negli studi legali: la costruzione e l'utilizzo di **prompt efficaci** (*legal prompting*).

In un sistema RAG (Retrieval-Augmented Generation), il prompt rappresenta il ponte tra la conoscenza umana e il motore linguistico.

Un prompt ben costruito consente di trasformare dati complessi in risposte utili, pertinenti e contestualizzate.

L'obiettivo è fornire strumenti teorici e pratici per aiutare l'Avvocato a comunicare efficacemente con un modello linguistico, migliorando la qualità dell'output e riducendo l'ambiguità.

Vedremo come formulare prompt giuridici ad alta precisione, perché funzionano e come adattarli ai diversi contesti normativi e procedurali dello studio legale.

Legal prompting: cosa rende efficace un prompt giuridico?

Un **prompt** è una richiesta strutturata che fornisce istruzioni testuali a un LLM (Large Language Model).

In ambito legale, il prompt ha il compito in particolare di:

- fornire contesto normativo o giurisprudenziale;
- delimitare lo stile e il formato della risposta;

- rendere espliciti gli obiettivi del professionista;
- garantire coerenza con il linguaggio forense.

Un prompt efficace in ambito giuridico deve essere:

- **Chiaro**: evitare ambiguità terminologiche e definire con precisione l'azione richiesta.
- **Contestualizzato**: includere norme di riferimento, fattispecie, o tipo di atto.
- **Controllabile**: strutturare il formato dell'output (lunghezza, linguaggio, sezioni).
- **Adattabile**: utilizzabile su casi simili con minime modifiche.

I prompt deboli o generici producono risposte generiche, spesso errate o poco utili. Un prompt ben progettato, invece, valorizza le potenzialità del modello, riducendo il bisogno di successive revisioni.

Come costruire prompt legali riutilizzabili con LangChain

LangChain è una libreria progettata per costruire flussi NLP flessibili e modulabili, anche in ambiente locale.

Nell'esempio seguente viene utilizzata la classe `OllamaLLM`, un'interfaccia specifica per eseguire LLM locali tramite Ollama, come ad esempio Mistral.

La sintassi adottata, tramite l'operatore `|`, consente di concatenare un `PromptTemplate` al modello scelto.

LangChain permette in particolare di:

- costruire **PromptTemplate** parametrizzati;
- concatenare prompt in sequenza (chaining);

- integrare LLM, memoria e vector store (FAISS).

Esempio: PromptTemplate dinamico

```
from langchain.prompts import PromptTemplate

from langchain_ollama.llms import OllamaLLM


# Definizione del template

template = """

Sei un avvocato specializzato in {ambito_legale}. Analizza il seguente testo:

{text}

Rispondi focalizzandoti su: {obiettivo}

"""


prompt = PromptTemplate(

    input_variables=["ambito_legale", "text", "obiettivo"],

    template=template,

)
```

```
# Inizializzazione del modello LLM locale via Ollama

model = OllamaLLM(model="mistral") # Richiede modello Mistral installato



# Creazione della catena LLM

chain = prompt | model



# Esecuzione con input specifico

data = {

    "ambito_legale": "diritto civile italiano",

    "text": '"Il locatore non risponde in alcun caso per danni derivanti da colpa grave

    nella manutenzione degli impianti dell'immobile locato.'",

    "obiettivo": "Verifica la clausola contrattuale"

}



response = chain.invoke(data)

print(response)
```

Spiegazione passo passo del codice:

1. **Importazione moduli**: si importano `PromptTemplate` e `OllamaLLM`, la classe che consente di utilizzare modelli locali con LangChain.
2. **Template del prompt**: si definisce una struttura per il prompt con tre variabili (`ambito_legale`, `text`, `obiettivo`) per generare richieste coerenti.
3. **Inizializzazione del modello**: `OllamaLLM` collega LangChain al modello locale specificato (in questo caso, `mistral`).
4. **Creazione della catena**: si usa l'operatore `|` per concatenare prompt e modello in modo leggibile.
5. **Esecuzione**: `chain.invoke()` permette di ottenere l'output passando un dizionario di input d'esempio con le variabili definite.

Questo approccio è pensato per essere leggero e modulare, ideale per sistemi locali basati su Ollama e Python, ed è facilmente integrabile nel flusso documentale di uno studio legale.

Permette di ottenere risposte giuridiche personalizzate a partire da testo strutturato, come un contratto o una sentenza.

Questo schema permette di riutilizzare un unico prompt adattabile a più ambiti (civile, penale, tributario…), mantenendo una struttura professionale e coerente.

LangChain consente anche di:

- aggiungere **memoria contestuale** tra richieste successive;
- usare **verifiche sui token** per evitare di superare la finestra del modello;
- **combinare output di prompt multipli** per generare documenti complessi.

Prompt giuridici spiegati. Legal prompting all’opera: esempi reali e analisi

Di seguito presentiamo prompt d’esempio adatti a casi comuni in uno studio legale.

Oltre al codice, spieghiamo **perché ciascun prompt funziona**, quali problemi risolve e come adattarlo.

a. Sintesi di documenti

Sistema: Riassumi in 500 parole il documento seguente, con attenzione a responsabilità e parte dispositiva.

Documento: {text}

Perché funziona: delimita la lunghezza, richiama due sezioni chiave (responsabilità e dispositivo), evita digressioni.

b. Confronto normativo

Sistema: Confronta l’art. [...] con l'art. [...]. Evidenzia differenze operative.

Perché funziona: il confronto è specifico e richiama norme note. Chiede differenze operative, non solo formali.

c. Redazione di pareri

Sistema: Redigi un parere su un caso di cittadinanza per matrimonio, con riferimento agli artt. 5, 6, 7 e 8 della legge 91 del 1992.

Fornisci struttura logica, riferimenti normativi e una conclusione motivata.

Fattispecie da esaminare: [...]

Perché funziona: simula la struttura tipica di un parere; guida il modello su struttura, riferimenti e finalità.

d. Analisi giurisprudenziale

Sistema: Analizza le sentenze allegate. Estrai principi consolidati in tema di onere della prova nel processo civile.

Perché funziona: focalizza l'attenzione su principi generali, evitando che il modello si perda nei dettagli del caso.

e. Classificazione automatica

Sistema: Classifica il documento seguente secondo queste categorie: atto giudiziario, contratto, normativa, corrispondenza, giurisprudenza.

Perché funziona: impone una tassonomia chiusa, migliorando l'efficacia nei processi documentali.

f. Verifica di riferimenti normativi

Sistema: Analizza il testo e individua gli articoli del codice civile citati. Riporta il numero dell'articolo e il contesto in cui è menzionato.

Perché funziona: aiuta a ricostruire il quadro normativo a partire da testi lunghi e spesso disorganizzati.

Strategie per migliorare i prompt nel tempo

Un prompt giuridico può (e deve) evolversi. Ecco alcune strategie pratiche:

- **A/B testing**: prova due versioni su casi simili e valuta quale produce risposte più accurate.
- **Check qualitativi**: costruisci una checklist per valutare completezza, correttezza giuridica e stile.
- **Versionamento**: conserva una cronologia delle revisioni dei prompt, associata agli output generati.
- **Prompt chaining**: suddividi attività complesse in più fasi (es. sintesi → confronto → suggerimento).

La progettazione di prompt non è statica: si affina con l'uso e il confronto con la pratica.

Cinque errori da evitare nella progettazione di prompt (con esempi)

- **Prompt vaghi**: "Spiegami questa sentenza" → troppo generico.
- **Domande doppie**: "Sintetizza la sentenza e dimmi se è corretta" → richiede due tipi di output diversi.
- **Nessun formato specificato**: "Scrivi un parere" → il modello può restituire testo discorsivo non strutturato.
- **Testo troppo lungo**: se supera i token, il modello, troncando, perde informazioni essenziali.
- **Contesto assente**: senza normativa o caso, l'output può essere incoerente o fuorviante.

Metodo R-CAFAR: schema pratico per formulare prompt legali efficaci

Per aiutare gli Avvocati a formulare richieste precise, ripetibili e controllabili ai modelli linguistici, proponiamo il metodo **R-CAFAR**, pensato specificamente per il contesto giuridico italiano:

R – Ruolo
Definisci il ruolo che l'IA deve assumere.

C – Contesto
Specifica l'ambito giuridico e il tipo di questione.

A – Attori e Fatti
Indica i soggetti coinvolti e i fatti rilevanti.

F – Finalità
Chiarisci l'obiettivo della richiesta.

A – Aggiornamento
Specifica se sono richiesti riferimenti aggiornati.

R – Richieste specifiche
Dettaglia ciò che vuoi ricevere: riferimenti normativi, sintesi, bozze, verifica di coerenza.

Esempio

Caso di assegno di mantenimento per figlio maggiorenne

R – Ruolo

Sei un avvocato esperto in diritto di famiglia italiano.

C – Contesto

Diritto di famiglia – determinazione dell'assegno di mantenimento per figlio maggiorenne non autosufficiente.

A – Attori e Fatti

Il figlio ha 24 anni, studia, non lavora e vive con la madre. Il padre vuole interrompere il mantenimento sostenendo che il figlio è adulto.

F – Finalità

Richiedo un parere legale motivato.

A – Aggiornamento

Normativa e giurisprudenza aggiornate al 2025.

R – Richieste specifiche

Analisi delle condizioni che giustificano il mantenimento, con sentenze recenti. Concludi con una valutazione sulla richiesta del padre.

Prompt completo:

Sei un avvocato esperto in diritto di famiglia italiano. In materia di assegno di mantenimento per figli maggiorenni, con aggiornamento al 2025, considera il caso di un ragazzo di 24 anni che studia all'università, vive con la madre e non ha reddito. Il padre si oppone alla prosecuzione dell'assegno, sostenendo che il figlio è adulto e può lavorare.

Puoi fornirmi un parere legale con riferimenti normativi e giurisprudenziali aggiornati, e concludere con una valutazione sulla legittimità della richiesta di revoca dell'assegno?

🧩 **Schema da ricordare**

R – Ruolo

-

C – Contesto

A – Attori e Fatti

F – Finalità

A – Aggiornamento

R – Richieste specifiche

Suggerimenti per lo studio legale

- **Centralizza i prompt migliori** in una cartella condivisa (con nome, data e ambito).
- **Assegna un referente AI** per la revisione e aggiornamento periodico dei prompt.
- **Testa sempre un prompt su un caso fittizio prima dell'uso operativo.**
- **Forma il personale** sui criteri base del prompting e sulle capacità del modello AI in uso.

Conclusioni

Un prompt efficace è una leva strategica per lo studio legale che adotta l'intelligenza artificiale.

Con strumenti come LangChain e LLM come Mistral eseguiti via Ollama, è possibile implementare un'interfaccia conversazionale locale d'esempio robusta, efficiente e controllabile.

Il prompt rappresenta il punto di partenza per costruire servizi di ricerca, sintesi, redazione di documenti e consulenza con l'ausilio dell'intelligenza artificiale.

Formare l'Avvocato all'uso corretto dei prompt è parte integrante dell'innovazione dello studio legale.

Il tema del legal prompting sarà ripreso in seguito per un ulteriore approfondimento.

11. Verificare la precisione dell’IA legale: strategie di testing e debug per sistemi RAG locali

Introduzione

Allucinazioni, bias, omissioni… Nel percorso di adozione dell’Intelligenza Artificiale generativa negli studi legali italiani, uno degli aspetti più delicati è garantire l’affidabilità delle risposte generate dai modelli linguistici.

Glossario di base:

- **LLM (Large Language Model)**: modello linguistico di grandi dimensioni addestrato su enormi quantità di testo per generare o comprendere linguaggio naturale.
- **RAG (Retrieval-Augmented Generation)**: tecnica che combina un motore di recupero documenti con un LLM per generare risposte basate su contenuti specifici.
- **Prompt**: testo di input fornito a un LLM per ottenere una risposta.
- **Embedding**: rappresentazione numerica di un testo che permette di confrontare la somiglianza semantica tra frasi o documenti.
- **FAISS**: libreria sviluppata da Facebook per creare e interrogare indici vettoriali ad alta velocità, utilizzata per il recupero di documenti.
- **LangChain**: framework Python per costruire applicazioni AI con modelli linguistici e catene di elaborazione.
- **Allucinazioni (Hallucinations)**: errore tipico degli LLM che consiste nella generazione di informazioni non supportate dai dati di origine.
- **Chunking**: tecnica di suddivisione del testo in segmenti più piccoli per facilitarne l’indicizzazione e il recupero.
- **Context window**: limite di memoria del modello linguistico, ovvero quanta informazione può tenere a mente in una sola richiesta.

Questo post, undicesimo della serie, affronta il tema del testing e del debug di un sistema RAG (Retrieval-Augmented Generation) installato in locale su ambiente Windows.

L'obiettivo è aiutare gli studi legali a costruire un processo robusto di verifica, utilizzando strumenti open source e metodologie replicabili su casi reali.

Perché testare un sistema RAG

Un sistema RAG, anche se locale e alimentato da dati propri, può produrre errori di vario tipo: per esempio risposte incomplete, allucinazioni, omissioni o bias.

Per uno studio legale, questi errori possono tradursi in conseguenze rilevanti sul piano professionale e deontologico.

Testare e migliorare le prestazioni del sistema è quindi una fase essenziale prima di introdurlo nel flusso di lavoro quotidiano.

Costruzione di un dataset di test

Per testare l'accuratezza di un sistema RAG, è utile costruire un dataset composto da:

- Documenti PDF legali già processati (testo pulito);
- Domande generate manualmente o automaticamente a partire dai documenti;
- Risposte attese (*ground truth*), redatte da un professionista.

Il dataset può essere conservato in un file JSON o CSV, con tre colonne: domanda, risposta attesa, contesto documentale.

Strumenti consigliati

RAGAS

RAGAS (Retrieval-Augmented Generation Assessment Suite) è una libreria Python progettata per valutare le prestazioni di un sistema RAG.

Fornisce metriche come “faithfulness”, “answer relevancy”, “context recall” e “context precision”.

Installazione:

pip install ragas

⚠️ Nota sulla riservatezza – Ragas

Ragas elabora i dati localmente e ***non trasmette prompt, risposte o documenti online****, salvo che tu non configuri servizi cloud esterni. Il software raccoglie solo statistiche d’uso anonime, disattivabili impostando* `RAGAS_DO_NOT_TRACK=true`.

LangSmith

LangSmith è uno strumento avanzato di LangChain che consente il tracciamento e l’ispezione delle catene di prompt.

Utile per il debug delle chiamate tra retriever, LLM e formattatori di risposta.

Installazione:

pip install langsmith

⚠️ **Nota sulla riservatezza – LangSmith**

LangSmith è uno strumento potente per il tracciamento delle pipeline, ma la versione cloud prevede la trasmissione dei dati verso server esterni gestiti da LangChain, anche fuori dall'Italia. Questo comporta che prompt, risposte generate e, in alcuni casi, anche il contesto documentale vengano inviati online per essere visualizzati e analizzati nella dashboard. Per queste ragioni, **LangSmith cloud è indicato solo in fase di sviluppo o test e non è adatto all'uso su dati legali reali, salvo specifiche garanzie contrattuali e valutazioni di impatto privacy**.

Tipi di errori da monitorare: allucinazioni, omissioni, bias, errori di retrieval

Durante l'utilizzo di un sistema RAG, è fondamentale saper riconoscere e classificare gli errori più comuni.

Questi errori possono compromettere la qualità delle risposte fornite e, di conseguenza, la fiducia nell'intero sistema.

Ecco una panoramica dettagliata:

1. **Allucinazioni** (**Hallucinations)** – Le allucinazioni si verificano quando il modello genera informazioni non supportate da alcun contenuto presente nei documenti di origine. Ad esempio, le allucinazioni possono attribuire una norma inesistente a una determinata legge, citare articoli di codice non coerenti con la giurisdizione italiana, oppure menzionare sentenze che non esistono. Gli errori dovuti alle allucinazioni sono tra i più pericolosi, in quanto possono indurre l'Avvocato a conclusioni erronee. Le cause delle allucinazioni possono risiedere in prompt

poco precisi, nella mancanza di contesto sufficiente o in LLM non ottimizzati per l'ambito legale.

2. **Incompletezza** – La risposta fornita è solo parziale rispetto alla domanda. Questo accade spesso quando il contesto fornito al modello non è sufficientemente ricco oppure quando la segmentazione (chunking) del documento originale è troppo aggressiva, escludendo parti rilevanti. In altri casi, può derivare da un limite del context window del modello o da un retrieval inefficace. È importante monitorare anche l'aderenza sintattica e semantica della risposta alla domanda originale.
3. **Bias** – I modelli linguistici possono riflettere pregiudizi culturali, sociali o ideologici presenti nei dati di addestramento. Sebbene l'utilizzo di modelli open source permetta un certo grado di controllo, è comunque importante monitorare la neutralità delle risposte, specialmente in ambito giuridico dove l'imparzialità è cruciale. Bias possono emergere ad esempio in risposte relative a temi sensibili come immigrazione, genere, minoranze, reati sessuali o religione. Una buona prassi è effettuare audit periodici su dataset specifici.
4. **Errori di retrieval** – Riguardano la fase in cui vengono selezionati i documenti da passare al modello per generare la risposta. Un retriever inefficace può scegliere fonti irrilevanti o marginali, influenzando negativamente la qualità dell'output finale. Questo può essere dovuto a una scelta subottimale della tecnica di embedding, a un'errata configurazione del motore FAISS o a una segmentazione poco funzionale. È utile valutare metriche di precisione/recall e verificare la qualità dei top-k documenti selezionati per ogni query.

Per ridurre al minimo tali errori è importante:

- Ottimizzare i prompt, specificando meglio il tipo di risposta desiderata (es. citazione normativa, sintesi, comparazione);
- Curare il pre-processing dei documenti (pulizia del testo, normalizzazione, segmentazione coerente);
- Aggiornare periodicamente gli indici FAISS e verificare l'efficacia degli embedding con set di validazione;
- Scegliere LLM adeguati al dominio legale e, se possibile, affinarli (*fine-tuning*) su dati giuridici specifici dello studio;

- Mantenere aggiornati i dataset normativi e giurisprudenziali utilizzati, per evitare che il sistema operi su testi superati o abrogati.

Inoltre, è raccomandabile:

- Effettuare un logging sistematico degli errori, associando ogni richiesta a un ID univoco e creando un archivio utile sia per il debug che per la formazione interna;
- Rafforzare la supervisione umana in tutte le fasi di test, validazione e utilizzo operativo: l'IA deve essere considerata **uno strumento di supporto e non sostitutivo del giudizio dell'Avvocato**, il quale mantiene sempre la **responsabilità legale e deontologica finale**.

Procedura consigliata per ambiente Windows

Si rimanda al post 6 per la configurazione dell'ambiente Windows.

Ricordiamo:

1. **Setup ambiente**: Windows 10 o superiore, Python ≥ 3.10, Ollama installato con modello compatibile (es. Mistral), FAISS attivo;
2. **Test iniziali manuali**: inserimento di domande via script Python e analisi delle risposte;
3. **Valutazione automatica** con RAGAS;
4. **Debug delle pipeline** con LangSmith;
5. **Verifica umana finale** su un sottoinsieme di risposte generate.

Best practice per studi legali

- Archiviare ogni risposta generata con il relativo riferimento documentale;
- Documentare sistematicamente gli errori riscontrati e le soluzioni adottate;
- Applicare versionamento sia al codice che ai dataset di test;
- Prevedere sempre la revisione umana delle risposte in caso di utilizzo operativo.

Context-Aware Generation (CAG)

Un aspetto cruciale nella valutazione della qualità delle risposte generate da un sistema RAG è il grado di consapevolezza del contesto, noto come **Context-Aware Generation (CAG)**.

Si tratta della capacità del modello linguistico di produrre risposte non solo grammaticalmente corrette, ma anche profondamente ancorate alle informazioni fornite nei documenti di riferimento.

Cos'è il CAG

Nel contesto dell'IA generativa applicata al diritto, il CAG implica che il modello linguistico utilizzato consideri:

- il significato specifico dei termini nel contesto giuridico (es. "revoca", "termine decadenziale");
- la coerenza tra la risposta e la giurisdizione o ambito normativo richiesto;
- i dettagli del caso o del documento fornito (es. se il documento è una sentenza o un parere).

Perché è importante

Una generazione sensibile al contesto è essenziale per:

- **evitare generalizzazioni** dannose in ambito giuridico;
- **prevenire allucinazioni** dovute a interpretazioni errate del contesto;
- **migliorare la rilevanza e precisione delle risposte**, specialmente nei casi in cui il sistema deve distinguere tra norme simili applicabili in ambiti differenti.

Come valutarla

Strumenti come **RAGAS** permettono di misurare il "context recall" e la "context precision", che indicano in che misura il contesto recuperato è stato utilizzato in modo corretto.

È inoltre possibile confrontare la risposta con il testo di origine per verificare l'aderenza semantica.

Implicazioni operative

Per ottenere un buon livello di CAG è consigliabile:

- Segmentare correttamente i documenti (chunking semantico, non solo strutturale);
- Sviluppare prompt che guidino il modello a *fare riferimento esplicito* al contesto;
- Monitorare i casi in cui la risposta omette, distorce o esagera parti del contesto.

Il CAG rappresenta un criterio avanzato ma sempre più essenziale per garantire che le risposte generate da un sistema RAG siano utilizzabili in un ambiente legale professionale, dove ogni parola può avere valore vincolante.

Conclusione

Testing e debug sono fasi imprescindibili per chi desidera integrare l'intelligenza artificiale nel proprio studio legale.

L'implementazione locale garantisce sicurezza e riservatezza, ma è essenziale verificarne l'accuratezza, correggere gli errori sistemici e costruire fiducia nei risultati.

L'adozione di strumenti come RAGAS rende questo processo più accessibile anche per professionisti senza competenze tecniche avanzate.

Nel prossimo post vedremo come estendere il semplice sistema RAG sperimentale che abbiamo creato.

12. IA e Avvocati: possibili estensioni del progetto

L’integrazione di sistemi avanzati di Intelligenza Artificiale (IA) negli studi legali sta trasformando il modo in cui Avvocati e personale legale gestiscono quotidianamente il flusso di informazioni e documenti.

In questo e nei prossimi articoli esploreremo alcune possibili estensioni innovative per il nostro progetto di implementazione di un sistema RAG (Retrieval-Augmented Generation) locale sperimentale.

Chatbot interno: un assistente virtuale sempre disponibile

Un chatbot interno basato sull’IA offre agli studi legali la possibilità di automatizzare e migliorare significativamente le interazioni quotidiane.

Integrando un chatbot con un sistema RAG, gli Avvocati possono:

- Ottenere rapidamente risposte precise a domande frequenti su normativa, giurisprudenza, precedenti casi trattati.
- Automatizzare la raccolta iniziale di informazioni dai Clienti, ottimizzando il tempo del personale.
- Offrire un primo livello di supporto in ambito interno, fornendo chiarimenti su procedure, norme interne e modelli documentali.

Come implementare un chatbot interno

L’implementazione pratica prevede l’utilizzo di strumenti open-source come Python, LangChain e Ollama, in combinazione con interfacce utente semplici ed intuitive realizzabili con librerie come Streamlit o Gradio.

Questo consente un rapido sviluppo e test del chatbot in ambiente Windows senza bisogno di grandi risorse hardware.

Query vocali: dialogare naturalmente con il sistema

L’introduzione delle query vocali migliora ulteriormente l’accessibilità e l’interattività dei sistemi di IA, permettendo agli utenti di interagire in maniera più naturale e immediata.

Tecnologie per le query vocali

Per implementare un sistema di query vocali è possibile utilizzare librerie Python come SpeechRecognition per catturare l’input vocale e pyttsx3 per la sintesi vocale. Questi strumenti possono essere facilmente integrati con il sistema RAG sperimentale già operativo, migliorando notevolmente l’esperienza utente.

- **SpeechRecognition** consente di convertire l’input vocale degli utenti in testo elaborabile dal sistema.
- **pyttsx3** offre funzionalità per restituire risposte tramite voce sintetizzata, creando una vera e propria conversazione vocale con il sistema.

Questa tecnologia permette agli Avvocati di interagire col sistema mentre lavorano su documenti o eseguono altre attività, riducendo al minimo le interruzioni del flusso di lavoro.

Gestione intelligente delle cartelle documentali

La gestione documentale è cruciale in ambito legale, dove l'organizzazione efficiente dei documenti può determinare l'efficacia dell'intero studio.

Sistemi di gestione documentale basati su IA

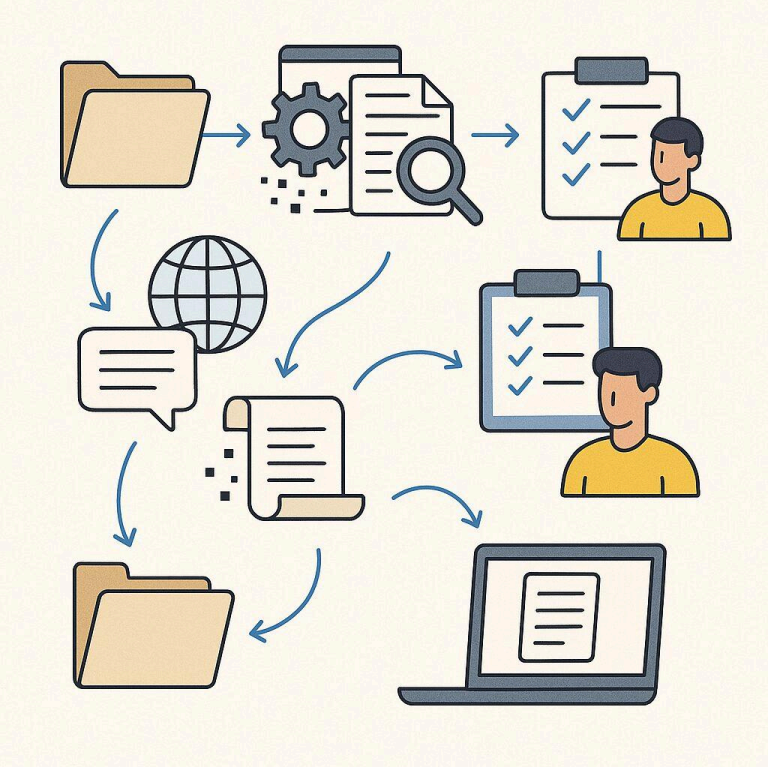
Implementare un sistema di gestione intelligente delle cartelle significa utilizzare algoritmi di intelligenza artificiale per categorizzare automaticamente i documenti, semplificare il reperimento delle informazioni e ridurre il rischio di errori manuali.

- Utilizzo di **Natural Language Processing (NLP)** per analizzare i testi legali, estrarre informazioni rilevanti (es. tipo di documento, parti coinvolte, riferimenti normativi) e taggare automaticamente i documenti.
- Impiego di modelli di **classificazione supervisionata** (es. Naive Bayes, SVM) e di **clustering** (es. KMeans, DBSCAN) per raggruppare logicamente i documenti anche in assenza di categorie predefinite.
- Conversione dei documenti in **embedding semantici** tramite modelli BERT o Sentence-Transformers, per creare ricerche testuali avanzate e cartelle virtuali dinamiche basate su significato e contesto.
- Integrazione con strumenti già esistenti attraverso script Python e interfacce semplificate per garantire la compatibilità con l'infrastruttura esistente.

Gestione documentale: flusso consigliato per l'automazione

1. **Acquisizione**: ogni nuovo documento inserito nel sistema viene automaticamente intercettato da un file watcher o da un'integrazione con email o sistemi di gestione documentale (DMS).
2. **Preprocessing**: tramite OCR (se necessario) e normalizzazione testuale si prepara il documento per l'elaborazione NLP.

3. **Classificazione e tagging**: il sistema analizza il contenuto del documento e assegna categorie, metadati e riferimenti rilevanti.
4. **Controllo di qualità e validazione**: un operatore può essere coinvolto per verificare o correggere il risultato del sistema di gestione documentale nei casi ambigui o a bassa confidenza.
5. **Organizzazione automatica**: il file viene salvato in una cartella coerente oppure indicizzato in una struttura virtuale (es. FAISS).
6. **Accesso intelligente**: l'utente può interrogare il sistema di gestione documentale in linguaggio naturale per trovare ciò che cerca, grazie a un motore di ricerca semantico o a un RAG.
7. **Audit e tracciabilità**: tutte le operazioni compiute sul documento possono essere registrate in log per garantire trasparenza e controllo.



Codice di esempio (semplificato) per clustering e tagging automatico

Prima di procedere con l'implementazione del codice di esempio per la gestione documentale, assicuratevi di aver completato la preparazione dell'ambiente Windows come descritto nel **post 6**, dove vengono illustrati i passaggi per installare Python, creare un ambiente virtuale e configurare l'ambiente di lavoro.

Per installare i pacchetti necessari, aprite il terminale nell'ambiente virtuale e digitate:

pip install scikit-learn

pip install numpy

pip install pandas

Questi pacchetti sono fondamentali per l'elaborazione del testo, la vettorializzazione e il clustering.

Una volta installati, potete utilizzare il seguente script come esempio per la gestione automatica dei documenti:

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

import os
```

```
# Carica documenti

docs = []

file_names = []

for filename in os.listdir("./documenti_legali"):

    with open(f"./documenti_legali/{filename}", "r", encoding="utf-8") as file:

        docs.append(file.read())

        file_names.append(filename)



# TF-IDF + clustering

vectorizer = TfidfVectorizer(stop_words=None)  # 'italian' non è supportato, si può usare None o una lista personalizzata

X = vectorizer.fit_transform(docs)

model = KMeans(n_clusters=4)

labels = model.fit_predict(X)



# Stampa etichette suggerite

for i, label in enumerate(labels):
```

```
print(f"{file_names[i]} --> Gruppo di appartenenza in base alla similarita' del contenuto testuale: cluster num. {label}")
```

Spiegazione passo passo:

Come eseguire lo script:

1. Salvare il codice in un file, ad esempio `clustering_documenti.py`
2. Aprire il terminale o prompt dei comandi, spostarsi nella cartella dove si trova il file
3. Eseguire lo script con il comando: `python clustering_documenti.py` Assicurarsi di avere la cartella `documenti_legali` con file `.txt` nella stessa directory.

Nota: questo codice di esempio è progettato per leggere solo file di testo (.txt) presenti nella cartella `documenti_legali`. Non funziona direttamente con file PDF, Word o immagini. Per supportare questi formati, è necessario integrare librerie specifiche.

1. **Importazione dei moduli:** si importano i componenti di Scikit-learn per la vettorializzazione del testo (`TfidfVectorizer`) e per il clustering (`KMeans`), oltre al modulo `os` per navigare nelle cartelle locali.
2. **Caricamento dei documenti:** lo script legge tutti i file di testo presenti nella cartella `./documenti_legali/`, li apre in lettura e salva sia il loro contenuto (`docs`) che i nomi dei file (`file_names`).
3. **Vectorizzazione TF-IDF:** ogni documento viene trasformato in un vettore numerico usando la tecnica TF-IDF, che misura l'importanza delle parole all'interno del corpus.
4. **Clustering con KMeans:** si applica l'algoritmo KMeans per dividere i documenti in 4 gruppi (cluster) in base alla similarità del contenuto testuale.

5. **Output dei risultati:** per ogni file, viene stampato a schermo il nome del file e il numero del cluster a cui appartiene, suggerendo un'organizzazione automatica dei documenti secondo criteri tematici impliciti.

Questo approccio può essere esteso per creare una *pipeline* completamente automatica, abbinando classificatori addestrati a *dashboard* per la revisione manuale dei risultati nei casi più complessi.

Il codice è una buona base didattica e pratica per il clustering di documenti legali in italiano. Per applicazioni professionali (ad esempio, organizzazione di archivi legali), si consiglia di:

- Integrare uno *stopwording* specifico per l'italiano,
- Gestire eventuali eccezioni sui file,
- Valutare il numero ottimale di cluster,
- Eseguire un preprocessing più raffinato.

Generatore di modelli legali personalizzati

Una delle attività più ripetitive all'interno di uno studio legale è la redazione di documenti standard: atti, contratti, istanze, diffide, lettere formali.

Automatizzare questa fase con un **generatore intelligente di modelli legali** consente di **standardizzare la produzione**, **ridurre gli errori** e **velocizzare il lavoro quotidiano**.

Il sistema si basa su **template dinamici** (es. in formato Word o Markdown) compilabili automaticamente in base a input forniti dall'utente (es. tipo di parte, data, clausole opzionali).

Può essere realizzato facilmente con Python usando il motore `Jinja2` per i modelli, e interfacce user-friendly come `Streamlit` o `Gradio` per la compilazione.

I dati possono essere inseriti manualmente o recuperati da moduli precedenti, integrando così un flusso coerente con il sistema documentale interno.

È una soluzione a basso impatto tecnico, ma di **altissimo rendimento operativo** per gli studi che redigono frequentemente atti simili.

Funzionalità e benefici

- **Velocità nella redazione**: Riduce drasticamente i tempi di preparazione dei documenti.
- **Uniformità**: Garantisce coerenza formale e sostanziale tra atti prodotti in studio.
- **Personalizzazione automatica**: Compila in automatico i dati ricorrenti (es. anagrafiche, riferimenti normativi, clausole standard) mantenendo la possibilità di modifiche manuali prima della stampa o dell'invio.

Tecnologie consigliate

- **Python + Jinja2**: per creare template dinamici HTML o TXT in cui inserire variabili prelevate da form o database.
- **Streamlit o Gradio**: per realizzare interfacce semplici in cui l'Avvocato può inserire i dati necessari (es. nome cliente, data udienza, oggetto del contratto).
- **LangChain + Ollama**: opzionalmente per la generazione del contenuto testuale a partire da prompt descrittivi o checklist giuridiche.

Esempio di flusso

1. **Scelta modello**: l'utente seleziona il tipo di documento da un elenco (es. contratto di locazione).

2. **Inserimento dati**: viene compilato un form con i dati del caso (parti, durata, importi, ecc.).
3. **Generazione automatica**: il sistema compila il template Jinja2 con i dati inseriti.
4. **Esportazione**: il documento viene reso disponibile in formato PDF, DOCX o TXT per revisione e invio.

Approfondiremo l'aspetto della gestione documentale in un post successivo.

Riepilogo automatico di atti giudiziari e contratti

Una delle estensioni più utili è la possibilità di generare automaticamente sintesi strutturate di atti giudiziari, contratti e altri documenti complessi.

Questa funzione, integrabile nel sistema RAG, consente:

- La creazione automatica di riepiloghi in linguaggio naturale, comprensibili anche da non tecnici;
- L'identificazione e l'evidenziazione di clausole chiave, riferimenti normativi e scadenze;
- L'indicizzazione delle sintesi per consentire ricerche rapide, confronti tra versioni e monitoraggio dei cambiamenti.

Tecnicamente, è possibile combinare modelli di linguaggio come quelli integrati con Ollama e LangChain con strumenti NLP avanzati (es. `spaCy`, `transformers`) per elaborare il testo e restituire un output sintetico, archiviabile e riutilizzabile.

Questa estensione rappresenta un passo avanti rispetto alla semplice risposta on-demand di un sistema RAG, in quanto permette allo studio legale di costruire nel tempo un vero e proprio archivio di schede riassuntive automatizzate, pronte per essere consultate o confrontate in ogni momento.

Anche questo aspetto sarà approfondito in un post successivo.

Fine-tuning: personalizzare il modello per compiti specifici

Un'ulteriore estensione del sistema RAG locale è rappresentata dal *fine-tuning* dei modelli linguistici.

Questa tecnica consente di adattare un modello pre-addestrato alle esigenze specifiche dello studio legale, migliorando la precisione e la pertinenza delle risposte generate.

Il fine-tuning è particolarmente utile quando:

- Si desidera che il modello comprenda e utilizzi terminologie legali specifiche.
- È necessario adattare il modello a stili redazionali o formati documentali propri dello studio.
- Si lavora con dati sensibili o proprietari che richiedono un trattamento personalizzato.
- Si intende migliorare le prestazioni del modello in compiti specifici, come la redazione di contratti o l'analisi di giurisprudenza.

Per implementare il fine-tuning in modo efficiente, è possibile utilizzare tecniche come LoRA (Low-Rank Adaptation), che permettono di aggiornare solo una piccola parte dei parametri del modello, riducendo significativamente il costo computazionale.

Strumenti come Hugging Face Transformers e PEFT (Parameter-Efficient Fine-Tuning) offrono le risorse necessarie per effettuare il fine-tuning su modelli open-source compatibili con Ollama, come Mistral o LLaMA.

L’integrazione del fine-tuning nel sistema RAG locale rappresenta un passo significativo verso la creazione di un assistente legale virtuale altamente specializzato, capace di comprendere e rispondere alle esigenze specifiche di un determinato studio legale.

Potenziali benefici per l’efficienza operativa

L’adozione di queste estensioni avanzate porta molteplici benefici concreti per l’organizzazione del lavoro legale, quali:

- **Efficienza operativa**: velocizza notevolmente la gestione delle informazioni, permettendo allo staff legale di concentrarsi su attività a più alto valore aggiunto.
- **Riduzione degli errori**: automatizzare processi di routine diminuisce significativamente la possibilità di errori umani.
- **Migliore organizzazione interna**: sistemi di classificazione e ricerca avanzata favoriscono l’ordine, la reperibilità e la tracciabilità documentale.

Si ricorda, in ogni caso, ancora una volta, che la **supervisione umana** resta imprescindibile: le soluzioni IA sono strumenti di supporto, non sostituti del giudizio professionale dell’Avvocato.

13. Giustizia predittiva: un esempio pratico

Introduzione

Glossario essenziale

- **Giustizia predittiva**: insieme di metodi e strumenti basati su intelligenza artificiale e machine learning, utilizzati per analizzare dati giuridici (come sentenze, atti e dati processuali) allo scopo di stimare con una certa probabilità l'esito di una controversia, la sua durata o altri parametri rilevanti. Non fornisce certezze né sostituisce il giudizio umano, ma offre supporto analitico alla valutazione strategica e decisionale.
- **Analisi predittiva**: tecnica basata su dati storici per stimare l'esito probabile di un evento futuro.
- **Machine learning (ML)**: ramo dell'IA che permette ai modelli statistici di "apprendere" dai dati senza programmazione esplicita.
- **Dataset**: insieme strutturato di dati organizzati in righe (casi) e colonne (variabili).
- **Modello predittivo**: algoritmo che riceve dati in input e produce una previsione o classificazione in output.
- **Random Forest**: algoritmo ML che combina molti alberi decisionali per aumentare precisione e robustezza.
- **Training/Test set**: suddivisione del dataset tra dati usati per "insegnare" al modello (training) e dati per valutarne l'efficacia (test).
- **Streamlit**: framework open-source per creare interfacce web interattive in Python, qui usato per simulare l'uso pratico del modello.
- **Explainability (spiegabilità)**: capacità di interpretare e giustificare il funzionamento e le decisioni di un modello.

L’adozione dell’Intelligenza Artificiale (IA) nello studio legale non è più un’ipotesi futuribile, ma una possibilità concreta per migliorare efficienza, strategia e servizio al Cliente.

In questo post presentiamo un semplice esempio pratico di **analisi predittiva delle controversie legali**, realizzato con strumenti open-source, funzionante **interamente in locale** e senza utilizzo di servizi cloud.

Tutto il sistema funziona localmente, senza invio di dati online.

Realizzeremo passo per passo:

- Un dataset simulato
- Un modello predittivo con Python
- Un’interfaccia leggera con Streamlit

Giustizia predittiva: obiettivi del progetto

- Comprendere cos’è la giustizia predittiva e come funziona con applicazioni concrete
- Fornire una base replicabile e modificabile per uno studio legale
- Usare una dashboard locale per analizzare un caso
- Introdurre strumenti di explainability
- Riflettere su limiti, opportunità e uso consapevole

Checklist per l’ambiente locale

Per la preparazione dell’ambiente di lavoro si rimanda al post 6.

Occorre installare da terminale, nell’ambiente virtuale del progetto:

pip install pandas scikit-learn streamlit joblib

Dataset simulato (anonimo)

Il dataset seguente rappresenta una simulazione semplificata di archivio processuale.

Ogni riga equivale a una controversia conclusa, con informazioni essenziali sul contesto e l’esito finale:

1 = esito favorevole

0 = esito sfavorevole

materia	foro	valore	tipo_parte	esito
lavoro	Roma	15000	persona fisica	1
civile	Milano	25000	impresa	0
recupero credito	Napoli	10000	persona fisica	1
lavoro	Napoli	12000	impresa	0
civile	Roma	22000	persona fisica	1
recupero credito	Milano	18000	impresa	0
lavoro	Milano	20000	persona fisica	1
civile	Napoli	14000	impresa	0

recupero credito	Roma	16000	persona fisica	1
lavoro	Roma	13000	impresa	0

Questo dataset è stato creato esclusivamente a scopo illustrativo. Non riflette alcuna base giurisprudenziale reale e **non ha alcun valore predittivo effettivo**.

Come generare il dataset simulato per il nostro esempio pratico di giustizia predittiva

Per creare il file `dati_controversie.csv` con i dati fittizi di esempio mostrati sopra, puoi utilizzare lo script Python seguente.

Verrà generato un file CSV pronto all'uso per il successivo addestramento del modello:

```
import pandas as pd


# Creazione del dataset simulato
dati = [
    ["lavoro", "Roma", 15000, "persona fisica", 1],
    ["civile", "Milano", 25000, "impresa", 0],
    ["recupero credito", "Napoli", 10000, "persona fisica", 1],
    ["lavoro", "Napoli", 12000, "impresa", 0],
```

```
    ["civile", "Roma", 22000, "persona fisica", 1],
    ["recupero credito", "Milano", 18000, "impresa", 0],
    ["lavoro", "Milano", 20000, "persona fisica", 1],
    ["civile", "Napoli", 14000, "impresa", 0],
    ["recupero credito", "Roma", 16000, "persona fisica", 1],
    ["lavoro", "Roma", 13000, "impresa", 0]
]

# Definizione delle colonne
colonne = ["materia", "foro", "valore", "tipo_parte", "esito"]

# Creazione e salvataggio del DataFrame
df = pd.DataFrame(dati, columns=colonne)
df.to_csv("dati_controversie.csv", index=False)
print("File 'dati_controversie.csv' creato con successo.")
```

Questo script può essere salvato come `crea_dataset.py` ed eseguito una sola volta per generare il file CSV con il dataset di esempio richiesto dal progetto.

Come deve essere costruito un dataset idoneo?

Per essere efficace in un sistema predittivo legale, un dataset deve possedere le seguenti caratteristiche principali:

- **Omogeneità dei campi**: ogni colonna deve rappresentare una variabile significativa (es. materia, foro, valore economico), con formato coerente.
- **Completezza**: ogni riga deve contenere tutti i dati richiesti, senza valori mancanti.
- **Pulizia**: eliminare errori, duplicazioni, o codifiche incoerenti (es. “milano” vs “Milano”).
- **Dimensione adeguata**: per semplici test didattici possono bastare 100–150 esempi. Tuttavia, per ottenere risultati affidabili in contesti reali, si raccomanda l’uso di almeno **500–1000 casi**, bilanciati tra esiti positivi e negativi, e arricchiti da variabili di qualità.
- **Etichettatura chiara**: la variabile `esito` deve essere costruita secondo criteri oggettivi (es. accoglimento/rigetto).
- **Neutralità**: evitare dataset dominati da una sola categoria o foro, per ridurre il rischio di generalizzazioni errate.

Nota: Il dataset utilizzato in questo post ha esclusiva finalità dimostrativa e non è strutturato secondo criteri statistici o giuridici validi per un uso professionale.

Non può quindi produrre risultati attendibili né essere impiegato per valutazioni reali.

La costruzione rigorosa del dataset è una condizione indispensabile per ottenere previsioni affidabili.

Senza una base informativa solida, completa e rappresentativa, nessun algoritmo — per quanto sofisticato — può restituire risultati utili o giuridicamente significativi.

Script Python con spiegazioni

Obiettivo generale del codice

Lo scopo dello script è costruire un semplice modello di machine learning di prova in grado di prevedere, con un certo margine di probabilità, l'esito di una causa legale a partire da quattro variabili descrittive: la materia trattata, il foro competente, il valore economico della causa e la tipologia della parte (impresa o persona fisica).

Questo processo viene gestito con l'aiuto della libreria `scikit-learn`, utilizzando un algoritmo noto come **Random Forest**, che funziona creando un insieme di alberi decisionali per arrivare a una previsione finale più robusta.

L'intero flusso è pensato per ambienti locali: niente cloud, nessun dato viene inviato all'esterno.

Funzionamento passo per passo

1. **Import delle librerie**
   Si importano le librerie Python necessarie per la gestione dei dati (`pandas`), l'addestramento del modello (`RandomForestClassifier`), la divisione dei dati (`train_test_split`), la valutazione (`accuracy_score`) e il salvataggio del modello (`joblib`).
2. **Caricamento del dataset**
   Viene letto il file `dati_controversie.csv` che contiene l'archivio simulato di cause concluse (il file deve essere presente nella medesima cartella del file .py del progetto). Ogni riga rappresenta un caso con le sue caratteristiche e l'esito.
3. **Pre-elaborazione dei dati (preprocessing)**
   Le colonne testuali (come `materia`, `foro`, `tipo_parte`) vengono trasformate

in formato numerico tramite `pd.get_dummies()` per essere comprese dal modello di machine learning.

4. **Separazione tra training e test set**
   Il dataset viene diviso in due parti:
   1. una per addestrare il modello (70%)
   2. una per testarne la capacità predittiva (30%)
5. **Addestramento del modello**
   Il classificatore Random Forest viene allenato con i dati di training. Il modello impara a riconoscere le correlazioni tra le caratteristiche della causa e il suo esito.
6. **Salvataggio del modello**
   Dopo l'addestramento, il modello viene salvato in un file `.pkl` che potrà essere riutilizzato successivamente dall'interfaccia utente (Streamlit) per effettuare previsioni su nuovi casi.
7. **Valutazione dell'accuratezza**
   Si confrontano le previsioni del modello sui dati di test con gli esiti reali per misurare l'efficacia del sistema. L'accuratezza viene stampata nel terminale.

Questo codice rappresenta una semplice base tecnica sulla quale può essere costruita una vera interfaccia predittiva a uso forense, che potrà evolvere nel tempo con l'aggiunta di parametri, filtri o modelli alternativi.

Il codice crea un modello di prova idoneo a prevedere l'esito di una controversia sulla base di 4 parametri.

Codice completo e commentato

```
import pandas as pd
```

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

import joblib



# 1. Caricamento dati

print("Caricamento...")

df = pd.read_csv("dati_controversie.csv")



# 2. Codifica variabili categoriche

X = pd.get_dummies(df.drop("esito", axis=1))

y = df["esito"]



# 3. Suddivisione training/test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)



# 4. Addestramento modello
```

```
clf = RandomForestClassifier()

clf.fit(X_train, y_train)



# 5. Salvataggio

joblib.dump(clf, "modello.pkl")



# 6. Valutazione

pred = clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, pred))
```

Riepilogo: il modello viene addestrato su casi passati per classificare i futuri.

Lo script deve essere salvato con estensione .py nella medesima cartella contenente il file del dataset simulato che abbiamo creato, ed eseguito da terminale.

Interfaccia Streamlit

Come funziona

Consente all'Avvocato di simulare un nuovo caso dal browser locale.

Codice

```
import streamlit as st

import pandas as pd

import joblib


clf = joblib.load("modello.pkl")


st.title("Analisi predittiva cause legali")


materia = st.selectbox("Materia", ["lavoro", "civile", "recupero credito", "Esci"])

if materia == "Esci":

    st.warning("Hai scelto di uscire. L'applicazione è stata interrotta.")

    st.stop()


foro = st.selectbox("Foro", ["Roma", "Milano", "Napoli"])

valore = st.number_input("Valore causa", min_value=5000)
```

```
tipo_parte = st.selectbox("Tipo parte", ["persona fisica", "impresa"])


input_df = pd.DataFrame([[materia, foro, valore, tipo_parte]], columns=["materia", "foro", "valore", "tipo_parte"])

input_encoded = pd.get_dummies(input_df).reindex(columns=clf.feature_names_in_, fill_value=0)


if st.button("Prevedi esito"):

    pred = clf.predict(input_encoded)[0]

    prob = clf.predict_proba(input_encoded)[0][1]

    st.success(f"Esito previsto: {'FAVOREVOLE' if pred == 1 else 'SFAVOREVOLE'}")

    st.info(f"Probabilità: {prob*100:.1f}%")

    st.warning("Questa è una stima probabilistica e non vincolante.")
```

Questo script richiede, nella medesima cartella che lo ospita, il file **“modello.pkl”**, generato eseguendo lo script che precede, di addestramento del modello.

Salva il codice come **app.py** ed eseguilo da terminale con il comando:

```
streamlit run app.py --server.address=localhost
```

L’interfaccia sarà accessibile solo sul tuo computer (**da browser**), all’indirizzo:

http://localhost:8501

Dopo essere uscito dal programma, per chiudere Streamlit **usa CTRL+C nel terminale o chiudi la finestra del terminale**.

Se stai usando un ambiente virtuale Python, ricordati di attivarlo prima.

Interpretabilità – SHAP e LIME

Installazione

pip install shap lime

Uso base

- **SHAP** mostra il contributo percentuale di ciascuna variabile
- **LIME** simula modifiche per spiegare localmente l’output

Esempio SHAP: `foro=Roma` ha contribuito al 24% alla previsione positiva.

Puoi integrare SHAP e LIME nel modello con esempi disponibili nella documentazione ufficiale.

Quadro normativo essenziale di riferimento per la giustizia predittiva

L’uso di strumenti predittivi e algoritmici nell’ambito del diritto è un tema che tocca da vicino principi costituzionali e norme sulla protezione dei dati personali.

Principale normativa di riferimento

- **GDPR (Regolamento (UE) 2016/679)**
- **Codice deontologico forense (in particolare artt. 9, 13, 14, 15, 24, 28, 35)**
- **Regolamento UE sull’intelligenza artificiale (Regolamento (UE) 2024/1689 (AI Act))**: classifica i sistemi predittivi in ambito giuridico come “ad alto rischio”.

Qualunque applicazione predittiva nello studio legale deve dunque essere trasparente, accessibile, verificabile e sempre utilizzata sotto il controllo dell’Avvocato (supervisione umana obbligatoria).

Per approfondire si rimanda al post 3.

Limiti e considerazioni etiche

- I modelli dipendono dalla qualità e dal bilanciamento dei dati
- Alcune variabili legali (per es. intenzioni, credibilità) sono non modellabili
- Attenzione a non cadere in una visione deterministica della giustizia

Es.: se il 90% dei dati proviene da Roma, il modello sarà sbilanciato a favore di quel foro.

Prospettive

Evoluzione del modello predittivo

L’algoritmo Random Forest è un ottimo punto di partenza per costruire un sistema stabile e interpretabile, ma in uno studio che accumula progressivamente nuovi dati reali è possibile:

- **Sostituire o affiancare modelli più avanzati**, come:
    - Gradient Boosting Machines (es. XGBoost, LightGBM), che spesso offrono una maggiore accuratezza.
    - Regressione logistica, utile per confronti con modelli lineari spiegabili.
    - Modelli di deep learning (es. reti neurali), se i dati crescono in volume e complessità.
- **Implementare un retraining continuo**, ad esempio ogni trimestre, in base ai nuovi casi archiviati, migliorando progressivamente le previsioni.
- **Creare ensemble ibridi** che combinano modelli diversi e bilanciano spiegabilità e accuratezza.

Questo approccio incrementale permette allo studio di non fermarsi a una soluzione fissa, ma di far evolvere il sistema al passo con la crescita del patrimonio informativo.

Automazione documentale e strategica

Oltre a fornire un risultato predittivo, il modello può diventare il motore per un flusso decisionale semi-automatizzato:

- **Generazione automatica di schede causa**: una volta ricevuta la previsione, si può creare una scheda riassuntiva con:
    - sintesi dei fattori considerati
    - stima del rischio
    - suggerimento strategico
- **Collegamento con modelli generativi (LLM)**: tramite modelli open-source, è possibile:

    - generare bozze di pareri motivati a partire dai parametri e dall'esito previsto
    - confrontare la previsione con orientamenti giurisprudenziali ricavati via RAG

In questo modo, l'output predittivo non rimane un dato a sé stante, ma il punto di partenza per un'intera catena decisionale.

Prospettive

- Integrazione con banche dati normative e giurisprudenziali
- Implementazione in RAG per generare spiegazioni giuridiche
- Report predittivi settimanali per cluster di fascicoli
- Interfaccia multi-utente per studi associati

Esempio: uno studio con 500 cause può usare un sistema predittivo per ordinare i casi in base al rischio e decidere dove investire tempo e risorse.

Conclusione

Il progetto presentato ha mera finalità esemplificativa: intende mostrare in modo pratico cos'è l'analisi predittiva applicata al diritto, come può essere implementata con strumenti accessibili, e quali opportunità offre in termini di supporto alla valutazione delle controversie.

L'integrazione tra conoscenza giuridica, metodo analitico e strumenti digitali locali permette in particolare di:

- **stimare rapidamente i rischi** di una causa,

- **documentare le scelte strategiche** con basi oggettive,
- **comunicare con il Cliente** in modo più trasparente.

Naturalmente, ogni decisione resta e deve restare nelle (insostituibili) mani dell'Avvocato.

Con il supporto dell'IA predittiva, la valutazione legale può tuttavia diventare più solida, misurabile e orientata all'efficacia.