Alma Mater Studiorum · Università di Bologna

Dipartimento di Informatica - Scienza e Ingegneria - DISI
Corso di Laurea in Informatica per il management

Applicazione AI-based per il supporto decisionale nella composizione e valutazione di team aziendali

Relatore:
Chiar.mo Prof.
Roberto Amadini

Presentata da:
Michela Manni

Sessione III
Anno Accademico 2024–2025

Indice

Introduzione

La gestione dei progetti aziendali rappresenta una delle sfide più complesse nel contesto dell'organizzazione del lavoro, in particolare per aziende operanti nel settore IT, dove la composizione dei team può incidere in modo significativo sull'esito dei progetti stessi. In questo contesto si inserisce il progetto sviluppato durante il tirocinio curricolare svolto presso OT Consulting S.r.l. (Object Technology Consulting), azienda di consulenza informatica che accompagna le imprese nella trasformazione digitale.

L'obiettivo principale del progetto è stato quello di realizzare un sistema in grado di suggerire, per ogni nuovo progetto, la composizione ottimale del team di lavoro. Questo suggerimento si basa sull'analisi di dati aziendali storici, utilizzando modelli di machine learning (apprendimento automatico) e tecniche avanzate di ricerca semantica su basi di dati vettoriali.

Il sistema sviluppato utilizza diverse componenti: un insieme di operazioni di estrazione e pulizia dei dati aziendali, modelli di classificazione per la predizione del successo progettuale, un agente AI per l'analisi automatica del project charter, una fase di ricerca dei progetti simili basata su Vector DB e infine una valutazione delle competenze dei dipendenti tramite Skill Matrix. L'insieme di queste converge in un'applicazione web, con interfaccia utente interattiva, in grado di supportare efficacemente il processo decisionale nella composizione dei team.

La presente tesi si propone di illustrare nel dettaglio il lavoro svolto, esponendo le soluzioni tecniche adottate, i risultati ottenuti e le riflessioni critiche emerse nel corso dello sviluppo. Dopo un primo capitolo introduttivo dedicato al contesto aziendale e agli obiettivi del progetto, si analizzeranno nel dettaglio le varie componenti del sistema e i modelli sperimentati. Infine, verranno esposti i risultati raggiunti e le prospettive di sviluppo futuro.

Capitolo 1

Contesto e Obiettivi

1.1 Il contesto aziendale

Il progetto è stato realizzato presso OT Consulting S.r.l., azienda con sede a Reggio Emilia attiva nel settore dell'information technology. OT Consulting offre soluzioni software personalizzate a clienti appartenenti a diversi settori, tra cui fashion e banking.

L'azienda possiede una base di dati storici relativi ai progetti software svolti, comprendenti informazioni rilevanti tra cui quelle sui membri dei team, le tecnologie adottate, le tempistiche e il livello di rischio. In questo scenario, la composizione dei team rappresenta un fattore cruciale per il successo dei progetti. Le decisioni in merito sono spesso complicate e si corre il rischio di optare per soluzioni non ottimali.

Da qui nasce l'idea di realizzare un sistema in grado di supportare tale processo decisionale, mediante l'analisi automatica dei dati storici aziendali.

1.2 Obiettivi del progetto

L'obiettivo principale del progetto è la realizzazione di un applicazione completa in grado di supportare la composizione dei team aziendali attraverso tecniche di intelligenza artificiale. In particolare, il sistema è progettato per:

- analizzare automaticamente i documenti di progetto (project charter), attraverso un agente AI basato su linguaggio naturale;
- identificare progetti passati simili, sfruttando la ricerca semantica su database vettoriali;
- valorizzare le competenze dei dipendenti tramite una skill matrix strutturata;
- suggerire proposte di composizioni ottimali di team per un nuovo progetto, integrando le informazioni precedenti;

- predire la probabilità di successo delle composizioni proposte, attraverso modelli di classificazione binaria e multiclasse;
- offrire un'interfaccia utente semplice, intuitiva e accessibile.

L'applicativo ha come obiettivo finale quello di fornire un valido supporto alle decisioni strategiche in ambito aziendale, fornendo raccomandazioni intelligenti e personalizzate nella selezione del team.

Capitolo 2

Architettura del Sistema

Il sistema sviluppato per la composizione dei team aziendali adotta un'architettura modulare, progettata per integrare diversi componenti indipendenti, ciascuno con responsabilità ben definite. La sezione seguente fornisce una descrizione ad alto livello dei componenti del sistema, la cui analisi dettagliata è approfondita nei capitoli successivi.

2.1 Panoramica generale

L'architettura è articolata nelle seguenti macro-componenti principali:

1. **Estrazione e preparazione dei dati**: operazioni per l'importazione, la pulizia e la strutturazione dei dati storici aziendali;
2. **Estrazione semantica dal project charter**: agente AI per l'estrazione dei dati rilevanti del nuovo progetto contenuti nel project charter;
3. **Modelli di classificazione**: algoritmi di machine learning per la predizione del successo progettuale;
4. **VectorDB e ricerca semantica**: sistema per il recupero dei progetti più simili attraverso rappresentazioni vettoriali;
5. **Skill Matrix**: meccanismo per la valutazione delle competenze tecniche dei dipendenti;
6. **Interfaccia utente**: applicazione web per l'interazione con l'utente.

2.2 Estrazione e preparazione dei dati

Il sistema parte dall'acquisizione dei dati grezzi aziendali, provenienti da file CSV. Tali dati contengono informazioni relative a progetti svolti, membri del team, tecnologie adottate, durata delle attività e indicatori di performance. Le informazioni più rilevanti contenute nei datasets verranno utilizzate in seguito per l'addrestamento dei modelli di machine learning.

Le operazioni di preprocessing sono implementate in Python tramite le librerie `pandas` e `NumPy` :

- rimozione di valori nulli, duplicati o incoerenti;
- normalizzazione delle colonne testuali (tecnologie, ruoli, ecc.);
- codifica delle variabili categoriche (one-hot encoding);
- generazione di feature derivate (es. nomi dei mesi lavorati).

2.3 Estrazione semantica dal project charter

Un aspetto importante del sistema è l'integrazione di un agente AI per l'analisi automatica del *project charter*. Il documento in questione contiene in formato testuale le informazioni relative al nuovo progetto, di cui si vuole comporre il team.

L'agente, sviluppato con la libreria LangChain, utilizza modelli linguistici pre-addestrati (GPT) per:

- segmentare il documento;
- identificare campi rilevanti;
- restituire una rappresentazione strutturata in formato JSON.

Questo output viene successivamente integrato grazie al formato che garantisce coerenza tra i progetti nuovi e quelli storici.

2.4 Moduli di classificazione

Una volta pre-elaborati i dati, essi vengono utilizzati per addestrare due tipologie di modelli di machine learning:

- **Modello binario**: classifica ogni progetto come "successo" o "insuccesso";

- **Modello multiclasse**: assegna un punteggio qualitativo basato su una metrica di marginalità aziendale.

I modelli sono addestrati con diversi algoritmi supervisionati (Random Forest, XGBoost, SVC, KNN, Gradient Boosting, Adaboost), valutati tramite *GridSearchCV* e selezionati sulla base delle metriche di performance (accuracy, precision, recall, f1-score). I modelli migliori sono serializzati con `joblib` per il riutilizzo in fase di predizione.

2.5 Ricerca semantica su VectorDB

Per generare le proposte di team suggeriti, il sistema utilizza una fase di ricerca semantica dei progetti simili. Ogni progetto, compreso quello nuovo, viene trasformato in un embedding vettoriale tramite il modello `all-MiniLM-L6-v2` di Sentence Transformers.

I vettori ottenuti vengono indicizzati nel database vettoriale **Milvus**, che consente il recupero efficiente dei progetti semanticamente più vicini.

La similarità tra progetti è un'informazione fondamentale per suggerire composizioni di team basate su esperienze pregresse. Il sistema estrae dai progetti più simili i team corrispondenti e li utilizza per generare le proposte di team.

2.6 Valutazione delle competenze: Skill Matrix

La matrice delle competenze, costruita a partire da un file Excel, associa ogni dipendente al proprio livello di competenza per ogni tecnologia, assegnando un punteggio da 0 a 4. Questa valutazione permette di:

- filtrare i membri che non possiedono skill adeguate;
- ordinare i candidati per affinità tecnica rispetto al progetto;

Questa fase permette al sistema di suggerire dipendenti competenti e qualificati nelle tecnologie necessarie allo sviluppo del nuovo progetto.

2.7 Interfaccia utente

L'interfaccia è realizzata con il framework **Gradio**, che consente di creare rapidamente applicazioni web in Python. L'utente è guidato in un flusso operativo a cinque fasi:

1. Caricamento dei datasets;
2. Upload del nuovo project charter;

3. Ricerca e visualizzazione dei progetti simili;
4. Caricamento della skill matrix;
5. Suggerimento dei team e predizione.

Capitolo 3

Tecnologie e strumenti utilizzati

Nel corso dello sviluppo dell'applicativo sono state utilizzate numerose tecnologie open source, selezionate in base all'affidabilità, alle prestazioni e alla loro integrazione con l'ecosistema Python. Questo capitolo presenta una panoramica degli strumenti impiegati, suddivisi per categoria funzionale.

3.1 Linguaggi e ambienti di sviluppo

Python

Il progetto è stato interamente sviluppato in **Python**, linguaggio molto diffuso in ambito data science e intelligenza artificiale grazie alla sua sintassi chiara e alla disponibilità di librerie specialistiche. Python è stato utilizzato sia per la logica di preprocessing e machine learning, sia per la costruzione dell'interfaccia utente.

PyCharm

Come ambiente di sviluppo è stato utilizzato **PyCharm**, un IDE professionale che ha facilitato la gestione del progetto, il controllo delle dipendenze, il debug e la navigazione tra i file. L'integrazione con ambienti virtuali Python ha permesso una gestione ordinata delle librerie installate.

3.2 Gestione dei dati e preprocessing

Pandas e NumPy

Per l'importazione e la manipolazione dei dati tabellari provenienti da file CSV ed Excel, sono state impiegate le librerie **Pandas** e **NumPy**. Questi strumenti hanno supportato

operazioni come il trattamento di valori mancanti, l'encoding delle variabili categoriche e la normalizzazione dei dati.

PyPDF e Openpyxl

La libreria **PyPDF** è stata utilizzata per il parsing dei *project charter* in formato PDF, mentre **Openpyxl** ha consentito l'accesso e l'elaborazione dei file Excel contenenti la skill matrix dei dipendenti.

3.3 Machine Learning e analisi del testo

Scikit-learn e XGBoost

Per l'addestramento e la valutazione dei modelli di classificazione sono state utilizzate le librerie **Scikit-learn** e **XGBoost**. La prima ha fornito gli strumenti per le operazioni di preprocessing e la validazione dei modelli, mentre XGBoost ha migliorato le prestazioni tramite tecniche di gradient boosting ottimizzato.

SentenceTransformers

La ricerca semantica è stata resa possibile tramite la libreria **SentenceTransformers**, che consente di convertire testi in vettori densi. In particolare, è stato utilizzato il modello preaddestrato `all-MiniLM-L6-v2`, adatto a task di semantic similarity.

LangChain e OpenAI

Per l'estrazione automatica di informazioni dai documenti testuali, è stato costruito un agente basato su **LangChain**, integrato con le API di **OpenAI** (modelli GPT). Questo modulo consente di strutturare automaticamente i dati contenuti nei project charter, convertendoli in JSON pronti per l'analisi.

3.4 Database vettoriale e ricerca semantica

Milvus

Il sistema di ricerca di progetti simili è stato realizzato utilizzando **Milvus**, un motore per database vettoriali open source altamente performante. I vettori semantici generati dalle informazioni dei progetti vengono memorizzati in Milvus e interrogati per similarità tramite il client Python ufficiale `pymilvus`.

3.5 Interfaccia utente

Gradio

L'interfaccia utente è stata realizzata con **Gradio**, un framework Python che permette la creazione di interfacce web interattive in pochi passaggi. L'utente è guidato attraverso un processo strutturato composto da cinque fasi: caricamento dei dati storici, parsing del project charter, ricerca dei progetti simili, caricamento della skill matrix e suggerimento del team ottimale. L'intero sistema è stato concepito per funzionare in locale.

FastAPI (modulo separato)

Parallelamente, è stato sviluppato un modulo a parte basato su **FastAPI**, con l'obiettivo di esporre i modelli ML tramite endpoint REST. Questo componente, attualmente separato dall'interfaccia principale, è stato pubblicato su **Render.com**.

3.6 Deployment e ambienti di esecuzione

Docker

Il sistema è stato containerizzato tramite **Docker**, strumento fondamentale per l'esecuzione di **Milvus** e per la creazione di ambienti replicabili in fase di sviluppo.

Render.com (per FastAPI)

Il servizio **Render.com** è stato utilizzato esclusivamente per il deployment sperimentale del modulo FastAPI, offrendo una soluzione semplice e scalabile per la pubblicazione di API su cloud. Tuttavia, l'applicativo principale (interfaccia Gradio) è progettato per l'esecuzione in locale.

Capitolo 4

Estrazione e Preprocessing dei Dati Aziendali

4.1 Origine e struttura dei dati

Il sistema si fonda sull'elaborazione di dati aziendali interni, resi disponibili in formato CSV ed Excel. Tali file contengono informazioni provenienti da due domini distinti: da un lato, l'insieme dei progetti software gestiti dall'azienda negli anni precedenti; dall'altro, la struttura organizzativa e le competenze tecniche del personale (Skill Matrix).

Il dataset relativo ai progetti include variabili quali il codice identificativo del progetto, la durata, le stime di effort, le tecnologie adottate, il cliente coinvolto, i mesi lavorati e il livello di rischio stimato. A ciascun progetto è inoltre associato un team di sviluppo, con indicazione del ruolo di ogni membro.

Parallelamente, la Skill Matrix è rappresentata da un file Excel che costituisce un inventario delle competenze tecniche dei dipendenti, espresso in forma tabellare, con un punteggio da 0 a 4 per ciascuna tecnologia.

Questa struttura eterogenea ha reso necessario lo sviluppo di una pipeline di estrazione e pulizia in grado di armonizzare i diversi formati e produrre un dataset coerente, utilizzabile per l'addestramento dei modelli e per la generazione delle raccomandazioni.

4.2 Preprocessing dei dataset storici

La prima fase ha riguardato l'elaborazione dei file CSV contenenti i dati relativi ai progetti. Le principali operazioni di preprocessing sono state implementate in linguaggio Python, con il supporto delle librerie `pandas` e `NumPy`. L'obiettivo era ottenere una base dati coerente, priva di errori e adatta all'addestramento dei modelli di machine learning.

I dati storici sono suddivisi in due dataset: il primo contiene le informazioni sui progetti conclusi, mentre il secondo riporta i dettagli relativi ai membri coinvolti. Entrambi

i dataset erano originariamente memorizzati dall'azienda in documenti Google e sono stati esportati in formato CSV per le elaborazioni successive.

Il primo dataset è composto da 357 righe e dalle seguenti colonne:

- **codice_epic**: codice identificativo del progetto, composto dal nome della procedura e da un numero univoco (stringa);
- **livello_di_rischio**: livello di rischio stimato dall'azienda per il progetto (`Low`, `Medium`, `High`, stringa);
- **total_worked_hours**: ore totali di lavoro effettuate (float);
- **total_worked_days**: giorni totali di lavoro, ottenuti dividendo le ore totali per 8 (float);
- **commercial_estimates**: stima commerciale da considerarsi in giorni(float);
- **delivery_estimates**: stima di consegna, pari all'80% della stima commerciale, anch'essa da considerarsi in giorni (float);
- **technologies**: elenco delle tecnologie impiegate (lista di stringhe);
- **assegnatario**: nome dell'assegnatario del progetto (stringa);
- **customer**: nome del cliente (stringa);
- **codice_offerta_OT**: identificativo interno aziendale (stringa);
- **creati**, **start_date**, **risolti**, **end_date**, **data_di_scadenza**: mesi di riferimento per creazione, avvio, chiusura, termine operativo e scadenza (stringhe);
- **success_unsuccess**: esito del progetto (1 = successo, 0 = insuccesso, boolean);
- **marginality**: indicatore numerico della marginalità del progetto (float).

Le colonne `success_unsuccess` e `marginality` sono state create appositamente per la realizzazione di questo progetto, come descritto nelle sezioni seguenti.

Il secondo dataset contiene invece 2187 righe e le seguenti colonne:

- **codice_epic**: codice del progetto associato, utilizzato per la relazione con l'altro dataset (stringa);
- **usr_name**: nome e cognome del dipendente che ha lavorato al progetto (stringa);
- **role**: ruolo ricoperto nel progetto (stringa);
- **hours**: ore lavorate dal singolo membro (float).

4.2.1 Definizione delle variabili target

Classificazione binaria

Per valutare l'esito dei progetti aziendali è necessario definire una metrica oggettiva che distingua i progetti completati con successo da quelli problematici. In questo lavoro, si è adottato un criterio basato sul confronto tra la durata effettiva del progetto e la stima di consegna concordata.

Un progetto è considerato di **successo** se viene completato in un tempo inferiore alla stima di consegna prevista, dimostrando efficienza operativa e rispetto delle scadenze. Al contrario, un progetto è classificato come **insuccesso** se richiede un tempo pari o superiore alla stima, indicando possibili inefficienze, sottostime iniziali o problematiche durante l'esecuzione.

Formalmente, la variabile binaria S (success) è definita come:

$$S = \begin{cases} 1 & \text{se } WD < DE \quad \text{(progetto completato in anticipo o nei tempi)} \\ 0 & \text{se } WD \geq DE \quad \text{(progetto in ritardo rispetto alla stima)} \end{cases} \tag{4.1}$$

dove:

- $S \in \{0, 1\}$: esito binario del progetto, con 1 che indica successo e 0 che indica insuccesso;
- WD (`total_worked_days`): numero complessivo di giornate lavorative effettivamente impiegate per completare il progetto. Questo valore è calcolato dividendo le ore totali lavorate per 8 (giornata lavorativa standard);
- DE (`delivery_estimates`): stima di consegna in giorni, calcolata convenzionalmente come l'80% della stima commerciale iniziale (CE). Questa riduzione riflette la prassi aziendale di definire una scadenza operativa più stringente rispetto alla promessa commerciale, lasciando un margine di sicurezza:

  $$DE = 0{,}8 \cdot CE$$

  dove CE (`commercial_estimates`) rappresenta la stima commerciale comunicata al cliente.

Questa formulazione, pur nella sua semplicità, consente di trasformare il problema in un compito di classificazione binaria supervisionata. La scelta di considerare come soglia la stima di consegna (DE) anziché la stima commerciale (CE) riflette le aspettative interne dell'azienda: un progetto che termina entro DE è considerato gestito efficacemente, mentre uno che supera tale soglia segnala criticità operative.

Classificazione multiclasse

Oltre alla classificazione binaria, è stato sviluppato un sistema di valutazione più articolato basato sulla **marginalità economica** del progetto, ossia la differenza relativa tra il tempo stimato e quello effettivamente impiegato. La marginalità M rappresenta una misura normalizzata dell'efficienza progettuale e viene calcolata come segue:

$$M = \begin{cases} 1 - \dfrac{WD}{CE} & \text{se } WD \leq DE \quad \text{(progetto efficiente)} \\ -\left(\dfrac{WD}{CE} - 1\right) & \text{se } WD > DE \quad \text{(progetto inefficiente)} \end{cases} \tag{4.2}$$

dove:

- $M \in R$: marginalità economica del progetto, espressa come frazione della stima commerciale;
- WD (`total_worked_days`): giorni effettivamente lavorati;
- CE (`commercial_estimates`): stima commerciale iniziale in giorni;
- $DE = 0{,}8 \cdot CE$ (`delivery_estimates`): stima di consegna operativa.

Nell'implementazione pratica, il rapporto $\frac{WD}{CE}$ viene troncato a 2 cifre decimali prima di essere utilizzato nel calcolo, e il risultato finale M viene anch'esso troncato a 2 decimali per garantire consistenza numerica.

Interpretazione della formula La formula distingue due modalità di comportamento progettuale:

1. **Caso 1: Progetto completato entro la stima di consegna** ($WD \leq DE$) In questo caso, il margine è calcolato come:

   $$M = 1 - \frac{WD}{CE}$$

   Esempio: considerando una stima commerciale pari a $CE = 100$ giorni e un completamento effettivo in $WD = 60$ giorni, la marginalità risulta:

   $$M = 1 - \frac{60}{100} = 0{,}40$$

   Un valore positivo indica che il progetto è stato eseguito con un consumo di tempo inferiore rispetto a quanto pianificato, generando un beneficio economico.

2. **Caso 2: Progetto che supera la stima di consegna** ($WD > DE$) In questo caso, il margine diventa negativo ed è calcolato come:

$$M = -\left(\frac{WD}{CE} - 1\right)$$

Esempio: per un progetto con stima commerciale $CE = 100$ giorni, stima di delivery $DE = 80$ giorni e un completamento in $WD = 120$ giorni, la marginalità è:

$$M = -\left(\frac{120}{100} - 1\right) = -0{,}20$$

Un valore negativo rappresenta uno sforamento dei tempi, che si traduce in una perdita economica.

Sebbene la variabile `marginality` sia continua, ai fini della classificazione multiclasse essa è stata discretizzata in quattro categorie ordinali mediante la funzione `pd.qcut()` di `pandas`, che suddivide i valori osservati in quartili:

- `Very bad`: quartile più basso (peggiori marginalità, forte sforamento);
- `Not so good`: secondo quartile;
- `Good`: terzo quartile;
- `Very good`: quartile più alto (migliori marginalità, massima efficienza).

Questa trasformazione consente di convertire il problema da regressione continua a classificazione supervisionata, dove ciascun progetto viene assegnato a una classe qualitativa rappresentativa del suo livello di successo economico.

La distribuzione delle istanze nelle quattro classi è riportata nella Tabella 5.8.

4.2.2 Pulizia-trasformazione dei dati

Sui dati raccolti sono state eseguite diverse operazioni di pulizia e trasformazione, sia manuali sia automatizzate, tra cui:

- eliminazione di record con valori nulli o incoerenti;
- codifica delle variabili categoriche tramite one-hot encoding;
- costruzione di feature derivate (es. mesi lavorati);

Le operazioni di correzione manuale (ad esempio la rimozione di valori nulli e la normalizzazione dei nomi) sono state effettuate direttamente nei documenti Google, mentre le procedure di trasformazione e codifica sono state implementate tramite codice Python.

4.2.3 Importazione-unione dei dataset

Entrambi i dataset sono stati scaricati dai Documenti Google come file CSV e importati tramite la libreria `pandas`. L'unione è stata realizzata mediante un'operazione di *merge* sulla chiave comune `codice_epic`, corrispondente all'identificativo univoco di ciascun progetto. Questa operazione ha permesso di accorpare le informazioni relative ai progetti con quelle dei membri che vi hanno partecipato, mantenendo soltanto i record coerenti presenti in entrambi i dataset.

4.2.4 Selezione-codifica delle feature

Dal dataset originale sono state rimosse le feature `codice_offerta_OT` e `role`, risultate debolmente correlate con la variabile target. Le date operative sono state sintetizzate nella variabile `mesi_lavorati`, contenente tutti i nomi dei mesi posizionati tra la data di inizio e la data di fine del progetto di riferimento.

Di seguito un riepilogo delle feature utilizzate per l'addestramento dei modelli insieme alle variabili target:

Tabella 4.1: Riepilogo delle feature finali utilizzate per l'addestramento dei modelli con le variabili target

Nome variabile	Tipo	Descrizione	Utilizzo
`livello_di_rischio`	Categoriale	Livello di rischio stimato dall'azienda (`Low`, `Medium`, `High`).	Feature
`commercial_estimates`	Numerica (float)	Stima commerciale del progetto.	Feature
`delivery_estimates`	Numerica (float)	Stima di consegna pari all'80% della stima commerciale.	Feature
`technologies`	Testuale (lista di stringhe)	Elenco delle tecnologie impiegate nello sviluppo.	Feature
`assegnatario`	Categoriale	Nome del referente o assegnatario del progetto.	Feature
`customer`	Categoriale	Nome del cliente associato al progetto.	Feature
`mesi_lavorati`	Testuale (lista)	Mesi compresi tra l'inizio e la fine del progetto.	Feature
`usr_name`	Testuale (stringa)	Nome del membro del team coinvolto.	Feature
`success_unsuccess`	Binaria (0/1)	Esito del progetto (1 = successo, 0 = insuccesso).	Target (class. binaria); Feature (class. multiclasse)
`marginality`	Numerica (float)	Marginalità del progetto, misura del livello di successo.	Target (class. multiclasse)

Codifica delle variabili categoriche

Le variabili non numeriche sono state convertite in formato numerico tramite **one-hot encoding**, applicato alle seguenti colonne:

- `technologies`;
- `mesi_lavorati`;

- `livello_di_rischio`, `assegnatario`, `customer`;
- `usr_name` (membri del team).

La procedura è stata implementata mediante la funzione `preprocess_data()`, che espande ogni colonna in una serie di feature binarie con la convenzione `colonna_valore` (es. `technologies_React`, `mesi_lavorati_giugno`). In particolare:

- la colonna `usr_name` è stata esplosa in più righe, generando una codifica binaria per ciascun membro;
- le colonne `technologies` e `mesi_lavorati` sono state binarizzate riga per riga, generando indicatori di presenza (1 se presente, 0 altrimenti);
- le variabili categoriali semplici (`livello_di_rischio`, `assegnatario`, `customer`) sono state trasformate in insiemi di variabili binarie.

Dopo l'espansione, le istanze multiple relative allo stesso progetto sono state aggregate tramite la funzione `groupby()` sulla chiave `codice_epic`, applicando un'aggregazione logica (`max`) sulle feature binarie. Questo approccio assicura la compatibilità dei dati con i modelli di machine learning, pur comportando un aumento significativo del numero di colonne generate.

4.3 Skill Matrix

Un secondo canale di acquisizione dei dati è rappresentato dalla Skill Matrix. Questa, fornita dall'azienda in formato Excel, rappresenta il livello di competenza di ciascun dipendente su un insieme di tecnologie rilevanti, con punteggi da 0 a 4. Dopo essere stata preprocessata, la matrice viene integrata nel sistema e utilizzata per confrontare le tecnologie richieste da ciascun progetto con le competenze effettivamente possedute dai candidati.

La logica di selezione basata sulle competenze sarà approfondita nel capitolo dedicato alla Skill Matrix.

4.4 Estrazione automatica dal project charter

Per acquisire in modo strutturato le informazioni relative al nuovo progetto, il sistema utilizza come input un documento PDF denominato *project charter*. Questo documento, redatto in linguaggio naturale, riassume le principali caratteristiche del progetto: cliente, tecnologie previste, valutazioni di rischio, stime di effort e date chiave. Solitamente ha una lunghezza di circa 5–6 pagine e dovrebbe essere redatto seguendo un template fornito

dall'azienda, che include sottotitoli esplicativi. Di seguito un esempio della prima pagina del Project Charter:

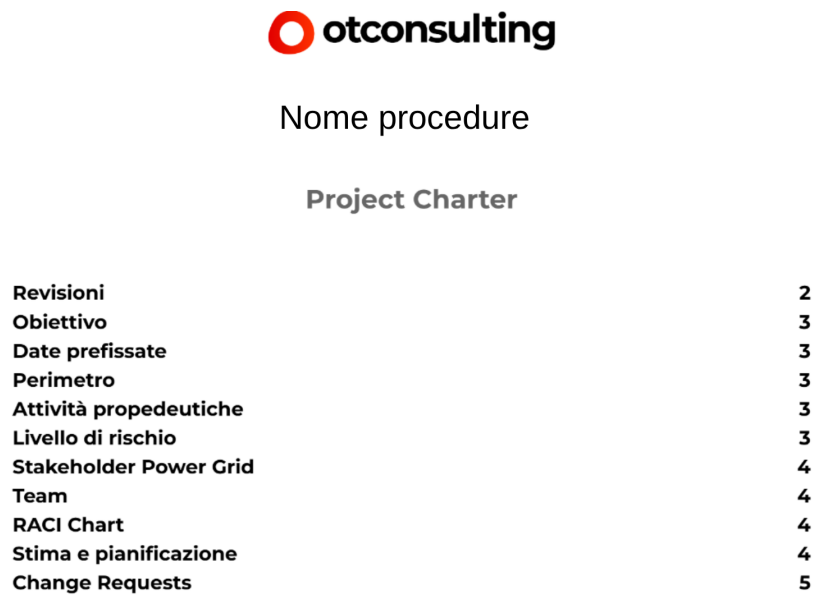

Figura 4.1: Indice tipico del Project Charter

Poiché il contenuto è testuale e non normalizzato, è stata sviluppata una componente intelligente in grado di analizzarne automaticamente il contenuto e restituire un output strutturato.

La componente in questione è denominata **Data Extraction Agent**, ed è basata su un agente LLM (Large Language Model) integrato con una catena di elaborazione sviluppata tramite la libreria `LangChain`. L'agente interroga il modello GPT-4 (nella versione `gpt-4o-mini`) fornito da OpenAI, seguendo un prompt costruito su misura per l'analisi semantica del documento.

Il prompt è progettato per guidare il modello nella ricerca e trasformazione di sei informazioni fondamentali:

- il **livello di rischio** del progetto, espresso in una delle seguenti categorie: `Low`, `Medium`, `High`, `Critical`;
- le stime numeriche relative all'effort: `commercial_estimates` e `delivery_estimates`, espresse come numeri decimali coerenti;
- la lista delle `technologies` indicate nel testo;
- il nome del `customer` o committente del progetto;

- l'elenco dei `mesi_lavorati`, calcolato a partire dalle date di inizio e fine.

Le istruzioni fornite al modello sono dettagliate e vincolanti, così da ridurre l'ambiguità interpretativa. Il prompt completo è riportato in **Appendice A**.

Il risultato dell'elaborazione viene poi processato da un `Output Parser` basato su `Pydantic`, che ne valida la struttura secondo uno schema JSON predefinito. Una volta validato, l'output viene salvato in formato JSON e utilizzato nelle successive fasi di ricerca e raccomandazione. Per garantire l'affidabilità dei dati estratti, le risposte generate dal modello sono state verificate manualmente, confrontandole con le informazioni presenti nel Project Charter.

L'utilizzo combinato di LLM, prompt engineering e parsing strutturato consente di automatizzare un'operazione altrimenti manuale e dispendiosa, rendendo il sistema scalabile e riutilizzabile per progetti futuri.

4.5 Integrazione con il sistema complessivo

Una volta preprocessati, i dati vengono salvati in formato JSON e resi disponibili per le componenti successive del sistema, che operano in modo coordinato ma indipendente. Il flusso di utilizzo dei dati avviene secondo la seguente logica:

- i modelli di machine learning vengono addestrati sui dati storici relativi ai progetti passati;
- la fase di ricerca semantica sfrutta le informazioni estratte dal nuovo project charter per individuare, all'interno del VectorDB, i progetti più simili; da questi vengono recuperati i team realmente utilizzati, che saranno proposti come composizioni iniziali;
- attraverso la Skill Matrix, il sistema individua i dipendenti più competenti nelle tecnologie richieste dal nuovo progetto; questi potranno essere suggeriti come sostituzioni o integrazioni ai team proposti;
- infine, i modelli di classificazione ricevono in input i dati del nuovo progetto abbinati ai diversi team suggeriti, e stimano per ciascuna configurazione la probabilità di successo o la marginalità attesa.

Questo flusso permette al sistema di supportare il processo decisionale in modo integrato, con un'interazione continua tra conoscenza storica, analisi semantica e valutazione predittiva.

L'intero processo è stato progettato per essere modulare, in modo da poter facilmente adattare o sostituire singole componenti senza compromettere il funzionamento globale del sistema.

Capitolo 5

Machine Learning per la Predizione del Successo Progettuale

Il presente capitolo descrive in dettaglio i modelli di apprendimento automatico sviluppati per supportare la fase di *team composition*, attraverso la predizione dell'esito atteso di un nuovo progetto.

Nel contesto della scienza dei dati e dell'intelligenza artificiale, i modelli di machine learning rappresentano strumenti fondamentali per la costruzione di sistemi intelligenti in grado di apprendere da dati storici al fine di effettuare previsioni, classificazioni o prendere decisioni autonome. Questi modelli si dividono in varie categorie, tra cui l'apprendimento supervisionato, non supervisionato e per rinforzo. In questa tesi ci si concentra sui modelli supervisionati, i quali apprendono una funzione a partire da un insieme di esempi etichettati (coppie input-output), con l'obiettivo di generalizzare su dati mai visti.

I modelli analizzati in questo lavoro appartengono alla categoria della classificazione e comprendono metodi basati su alberi decisionali, nonché modelli lineari e basati sulla distanza. Ciascuno di questi modelli presenta vantaggi e limitazioni in termini di accuratezza, interpretabilità, robustezza e tempo di addestramento.

La classificazione viene affrontata con due strategie distinte:

- **Classificazione binaria**, che prevede l'etichettamento del progetto come "successo" o "insuccesso";
- **Classificazione multiclasse**,che assegna un progetto a una delle classi ordinali definite a partire dalla marginalità (*Very bad*, *Not so good*, *Good*, *Very good*), ottenute dalla discretizzazione in quartili del margine economico dell'iniziativa.

Entrambe le tipologie di classificazione sono state implementate in Python, utilizzando `scikit-learn`, `XGBoost` e altre librerie correlate. In questo capitolo si presentano i

dati utilizzati, le tecniche di preprocessing, la struttura degli algoritmi, la ricerca degli iperparametri e l'analisi dei risultati.

5.1 Modelli

Nel progetto presentato, i modelli di machine learning predicono l'esito atteso di un progetto, sulla base di un insieme di dati strutturati e pre-elaborati. L'approccio seguito è supervisionato: i modelli vengono addestrati a partire da esempi etichettati, ovvero osservazioni storiche per le quali è noto l'esito (binario o multiclasse), al fine di apprendere una funzione che generalizzi su dati futuri.

Le tecniche adottate rientrano tutte nella categoria della classificazione, ovvero quelle in cui il compito consiste nell'assegnare un'osservazione a una tra più categorie discrete. Per tale ragione, non si è fatto ricorso a modelli di regressione o ad approcci non supervisionati.

Gli algoritmi selezionati sono stati scelti in base alla loro affidabilità, flessibilità e disponibilità nelle principali librerie open source Python, tra cui scikit-learn, XGBoost, e Imbalanced-learn.

Nello specifico, sono stati impiegati i seguenti algoritmi di classificazione:

- **Random Forest**;
- **XGBoost**;
- **K-Nearest Neighbors**
- **Gradient Boosting**
- **AdaBoost**
- **Support Vector Classifier (SVC)**.

Tutti i modelli sono stati utilizzati sia per la classificazione binaria che multiclasse, a eccezione di KNN, Gradient Boosting e AdaBoost, impiegati solo nella variante binaria per motivi legati alle performance sperimentali osservate. Le descrizioni che seguono illustrano il funzionamento teorico di ciascun algoritmo.

5.1.1 Random Forest

Random Forest è un algoritmo di tipo ensemble introdotto per mitigare i limiti degli alberi decisionali tradizionali, limitati dalla loro tendenza all'overfitting. L'idea di base è quella di costruire una "foresta" composta da più alberi decisionali, ciascuno dei quali

viene addestrato su un sottoinsieme casuale del dataset, selezionato tramite campionamento con rimpiazzo (bootstrap). In aggiunta, a ogni split interno degli alberi, viene considerato solo un sottoinsieme casuale delle feature disponibili.

Questa doppia casualità (sui dati e sulle feature) permette di ottenere alberi diversificati, i cui errori tendono a compensarsi nel processo decisionale finale, che avviene tramite votazione di maggioranza nel caso della classificazione. Il modello risultante è quindi più stabile, meno soggetto a fluttuazioni nei dati e più generalizzabile.

Il modello Random Forest può essere impiegato sia per compiti di classificazione sia di regressione, e supporta nativamente strategie di bilanciamento classi, gestione dei dati mancanti e stima dell'importanza delle feature. La classe RandomForestClassifier fornisce un'interfaccia completa per l'uso di questo algoritmo, permettendo il controllo fine su parametri come il numero di alberi (n_estimators), la profondità massima (max_depth) e il numero di feature selezionate per split (max_features) [7].

5.1.2 XGBoost

XGBoost (Extreme Gradient Boosting) è un potente algoritmo basato sul principio del boosting graduale, in cui una sequenza di modelli viene costruita in modo incrementale, con ciascun nuovo modello che cerca di correggere gli errori residui dei precedenti. A differenza del Random Forest, che lavora in parallelo sugli alberi, XGBoost li costruisce in maniera sequenziale, ottimizzando a ogni passo una funzione obiettivo che include sia una componente di perdita (per esempio log-loss) sia un termine di regolarizzazione.

L'algoritmo è progettato per essere estremamente efficiente e scalabile. Integra tecniche avanzate come la pruning anticipata (early stopping), regolarizzazione L1 e L2, gestione dei valori mancanti e ottimizzazione basata su grafi. L'implementazione in Python tramite la classe XGBClassifier consente l'utilizzo dell'algoritmo in contesti di classificazione con un controllo granulare sugli iperparametri.

Grazie alla sua struttura modulare, XGBoost può essere impiegato efficacemente in scenari con molte feature, dati eterogenei o con forte disomogeneità tra classi. Proprio per questa sua versatilità, è spesso utilizzato in contesti produttivi e in competizioni internazionali di machine learning [13].

5.1.3 K-Nearest Neighbor

Il K-Nearest Neighbors è un algoritmo intuitivo che appartiene alla categoria dei classificatori "lazy", ovvero che non costruiscono un modello esplicito durante la fase di addestramento. Il principio di base è che un'istanza viene classificata sulla base delle classi osservate tra i k esempi più vicini, individuati tramite una metrica di distanza predefinita.

Il modello KNeighborsClassifier può impiegare diverse metriche di distanza (come Euclidea, Manhattan, Chebyshev) ed è particolarmente sensibile alla scala delle feature, motivo per cui è spesso abbinato a tecniche di normalizzazione o standardizzazione.

Il KNN si presta bene a problemi in cui le classi presentano confini non lineari, ma può risultare computazionalmente oneroso all'aumentare del numero di osservazioni. Inoltre, la scelta del parametro k ha un impatto diretto sulla capacità del modello di generalizzare correttamente [10].

5.1.4 Gradient Boosting

Il Gradient Boosting è una tecnica ensemble basata sull'addestramento progressivo di modelli deboli, generalmente alberi decisionali a bassa profondità. A ogni iterazione, l'algoritmo costruisce un nuovo albero che cerca di correggere gli errori residui accumulati dal modello corrente, minimizzando la funzione di perdita tramite discesa del gradiente.

La classe GradientBoostingClassifier implementa questo approccio in modo efficiente, consentendo di specificare parametri fondamentali come il numero di stadi (n_estimators), il learning rate (learning_rate) e la profondità massima degli alberi (max_depth).

Una delle principali caratteristiche del Gradient Boosting è la sua capacità di catturare relazioni non lineari tra le feature, pur mantenendo un alto grado di controllo sul rischio di overfitting grazie a meccanismi di regolarizzazione e subsampling [6].

5.1.5 AdaBoost

AdaBoost (Adaptive Boosting) è uno degli algoritmi di boosting più noti e semplici da implementare. Il suo funzionamento si basa sull'addestramento sequenziale di una serie di modelli deboli, spesso alberi decisionali molto semplici (depth=1, chiamati "stumps"), dove ogni nuovo modello cerca di correggere gli errori del precedente attribuendo maggiore peso agli esempi più difficili.

L'algoritmo è disponibile tramite la classe AdaBoostClassifier, che consente di specificare il numero di modelli (n_estimators), il tasso di apprendimento (learning_rate) e il tipo di classificatore base utilizzato.

AdaBoost è particolarmente efficace in presenza di dati semplici e ben separabili, ma può soffrire in contesti molto rumorosi, dove tende a sovra-adattarsi agli outlier. Nonostante ciò, rappresenta una soluzione leggera e facilmente interpretabile per problemi di classificazione supervisionata [5].

5.1.6 Support Vector Classifier (SVC)

Il Support Vector Classifier, basato sulla teoria delle macchine a vettori di supporto (SVM), mira a trovare un confine decisionale ottimale tra le classi, massimizzando il margine tra gli esempi più vicini ai bordi del separatore. In scenari non linearmente

separabili, l'algoritmo può impiegare funzioni kernel per proiettare i dati in spazi di dimensione superiore dove la separazione è possibile.

SVC supporta diversi tipi di kernel (lineare, polinomiale, RBF) e offre una gestione flessibile della regolarizzazione attraverso il parametro C, che controlla il compromesso tra ampiezza del margine e correttezza della classificazione.

L'uso del kernel trick rende il modello particolarmente potente, ma anche più sensibile alla scelta dei parametri, richiedendo quindi attenzione in fase di tuning. SVC è noto per le sue prestazioni eccellenti su dataset di piccole o medie dimensioni, soprattutto quando le feature sono ben scalate e il numero di dimensioni non è eccessivo [11].

5.2 Metriche di valutazione

Per valutare le prestazioni dei modelli di classificazione sviluppati, sono state utilizzate le metriche standard fornite da `scikit-learn` [8, 9].

- **Accuracy**: misura la proporzione di osservazioni classificate correttamente rispetto al totale. È definita come:

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

  Dove:

  - TP (True Positive): numero di istanze positive classificate correttamente;
  - TN (True Negative): numero di istanze negative classificate correttamente;
  - FP (False Positive): numero di istanze negative classificate erroneamente come positive;
  - FN (False Negative): numero di istanze positive classificate erroneamente come negative.

- **Precision**: indica la percentuale di predizioni positive corrette, e si calcola come:

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

  Un valore elevato di precision indica che il modello commette pochi falsi positivi.

- **Recall (Sensibilità)**: rappresenta la capacità del modello di identificare correttamente le istanze positive:

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

  Un valore elevato di recall implica che il modello commette pochi falsi negativi.

- **F1-score**: è la media armonica tra precision e recall, utile per bilanciare le due metriche:
$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$
Questa metrica è particolarmente indicata in caso di dataset sbilanciati, in quanto combina in un singolo valore la capacità del modello di evitare sia falsi positivi sia falsi negativi.

- **Support**: indica il numero di campioni appartenenti a ciascuna classe nel dataset di test. È utilizzato per calcolare le medie ponderate delle metriche e per interpretare la distribuzione delle classi.

Il classification report di `scikit-learn` riporta queste metriche per ciascuna classe presente nel dataset. Ogni classe, anche nel caso della classificazione binaria, viene considerata a turno come positiva, così da calcolare precision, recall e F1-score in modo specifico per tutte le categorie. Questa scelta permette di valutare con maggiore dettaglio il comportamento del modello sulle singole classi, evidenziando eventuali differenze nella capacità di riconoscere correttamente successi e insuccessi. L'inclusione del support facilita inoltre l'interpretazione dei risultati, indicando la distribuzione reale delle osservazioni in ciascuna classe.

5.3 Classificazione Binaria

5.3.1 Definizione del target

L'obiettivo della classificazione binaria è predire se un progetto avrà successo, secondo la seguente regola:

$$S = \begin{cases} 1 & \text{se } WD < DE \\ 0 & \text{se } WD \geq DE \end{cases} \tag{5.1}$$

Dove WD rappresenta i giorni lavorati e DE la stima di consegna.

Per la classificazione binaria vengono utilizzati tutti i modelli precedentemente analizzati.

5.3.2 Suddivisione del training e test set

Per valutare correttamente le prestazioni dei modelli di machine learning è fondamentale separare i dati disponibili in due sottoinsiemi distinti:

- **Training set**: utilizzato per addestrare i modelli, ovvero per far apprendere i pattern e le relazioni tra le feature e la variabile target;

- **Test set**: mantenuto completamente separato durante l'addestramento e utilizzato esclusivamente per valutare le prestazioni del modello su dati mai visti, simulando l'applicazione reale su nuovi progetti.

Questa separazione è cruciale per evitare il fenomeno dell'*overfitting*, in cui un modello memorizza i dati di training anziché apprendere regole generalizzabili, risultando poi inefficace su nuove istanze.

La suddivisione del dataset in training e test set è stata effettuata mediante una funzione dedicata, riportata di seguito:

```
def prepare_data(df):
    X = df.drop(columns=['success_unsuccess'], errors='ignore')
    y = df['success_unsuccess']
    return train_test_split(
        X, y, test_size=0.3, random_state=42, shuffle=True, stratify=y
    )
```

La funzione `prepare_data()` riceve in input il *DataFrame* pre-processato e separa le variabili indipendenti (`X`) dalla variabile target (`y`), identificata come `success_unsuccess`. Successivamente, il dataset viene suddiviso in due sottoinsiemi: il *training set* (70% dei dati) e il *test set* (30%). Il parametro `random_state=42` garantisce la riproducibilità dei risultati, mentre l'opzione `stratify=y` assicura che la distribuzione delle classi rimanga proporzionale in entrambe le partizioni. L'opzione `shuffle=True` consente inoltre di mescolare casualmente i campioni prima della suddivisione, riducendo possibili bias legati all'ordine dei dati.

5.3.3 Singoli Modelli

Per la classificazione binaria sono stati utilizzati utilizzati tutti i modelli precedentemente analizzati. Di seguito si riportano l'analisi degli iperparametri selezionati per ciascun modello e la relativa valutazione delle prestazioni.

Random Forest

L'intero modello è stato ottimizzato tramite **RandomizedSearchCV**, una strategia di ricerca casuale nello spazio degli iperparametri, più efficiente della grid search in presenza di un numero elevato di combinazioni. Per la validazione è stato impiegato uno schema **StratifiedKFold** con 5 suddivisioni, che assicura la conservazione del bilanciamento tra classi in ciascun fold.

Di seguito si riportano gli iperparametri ottimizzati e la loro funzione:

- `clf__n_estimators`: numero di alberi nella foresta. Valori testati: 100, 200, 300, 500. Un numero maggiore di alberi tende a migliorare l'accuratezza e la stabilità del modello, al costo di un maggiore tempo computazionale.

- `clf__max_depth`: profondità massima degli alberi. Valori: `None` (illimitata), 10, 20, 30. Serve a limitare l'overfitting controllando la complessità degli alberi.
- `clf__min_samples_split`: numero minimo di campioni richiesti per suddividere un nodo. Valori: 2, 5, 10. Impedisce la creazione di nodi troppo piccoli e instabili.
- `clf__min_samples_leaf`: numero minimo di campioni richiesti per un nodo foglia. Valori: 1, 2, 4. Parametri più alti portano a modelli più semplici e generalizzabili.
- `clf__max_features`: numero massimo di feature da considerare per la suddivisione a ogni nodo. Valori: `'auto'`, `'sqrt'`, `'log2'`, 0.2, 0.5. Influenza la diversità tra gli alberi: valori più bassi aumentano la varietà, contribuendo all'effetto ensemble.
- `clf__bootstrap`: specifica se utilizzare il campionamento con rimpiazzamento (bootstrap). Valori: `True`, `False`. Il bootstrap incrementa la diversità degli alberi, ma in certi casi l'uso senza rimpiazzamento può dare risultati migliori.
- `clf__criterion`: funzione utilizzata per misurare la qualità della suddivisione. Valori: `'gini'` e `'entropy'`. Entrambe sono misure dell'impurità: `gini` è più veloce, mentre `entropy` è più informativa in certe situazioni.

Nel complesso, questa architettura combinata ha permesso di costruire un classificatore robusto, capace di affrontare sia lo squilibrio tra le classi, sia l'elevata dimensionalità del dataset.

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Insuccesso (0)	0.70	0.68	0.69	56
Successo (1)	0.62	0.64	0.63	45
Accuracy		0.66		

Tabella 5.1: Valutazione del modello Random Forest (una singola esecuzione)

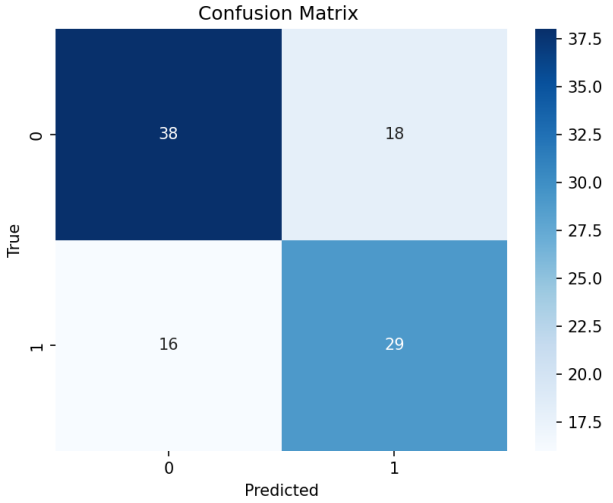

Figura 5.1: Matrice di confusione — Random Forest (una singola esecuzione)

XGBoost

Nel contesto di questo progetto, XGBoost è stato combinato con **SMOTE** all'interno di una pipeline (`ImbPipeline`) per affrontare lo squilibrio tra le classi. Il dataset è stato prima bilanciato artificialmente tramite la generazione sintetica di campioni minoritari, e successivamente utilizzato per l'addestramento del classificatore.

La fase di ottimizzazione degli iperparametri è stata condotta tramite `RandomizedSearchCV`, con validazione stratificata a 5 fold. Di seguito vengono descritti gli iperparametri utilizzati:

- `clf__n_estimators`: numero di alberi da costruire. Valori testati: 100, 200, 300, 500. Un numero maggiore può aumentare la capacità predittiva, ma anche il rischio di overfitting.
- `clf__max_depth`: profondità massima degli alberi. Valori: 3, 6, 9, 12. Profondità maggiori permettono di modellare interazioni complesse, ma aumentano il rischio di overfitting.
- `clf__learning_rate`: tasso di apprendimento. Valori: 0.01, 0.05, 0.1, 0.2. Riduce l'impatto di ogni albero successivo, rallentando l'apprendimento ma migliorando la generalizzazione.
- `clf__subsample`: percentuale di campioni da utilizzare per ciascun albero. Valori: 0.6, 0.8, 1.0. Aiuta a prevenire overfitting introducendo casualità nel training.

- `clf__colsample_bytree`: percentuale di feature da considerare per ogni albero. Valori: 0.6, 0.8, 1.0. Migliora la diversità degli alberi e la generalizzazione.
- `clf__gamma`: soglia minima di gain per consentire una divisione. Valori: 0, 0.1, 0.2, 0.3. Valori maggiori rendono il modello più conservativo, evitando split poco informativi.
- `clf__min_child_weight`: peso minimo richiesto in un nodo foglia. Valori: 1, 3, 5. Valori più alti evitano la creazione di foglie con pochi campioni.
- `clf__reg_alpha`: termine di regolarizzazione L_1 (lasso). Valori: 0, 0.01, 0.1. Penalizza le caratteristiche meno rilevanti, favorendo la sparsità del modello.
- `clf__reg_lambda`: termine di regolarizzazione L_2 (ridge). Valori: 1, 1.5, 2.0. Stabilizza l'ottimizzazione penalizzando grandi coefficienti.

Infine, il parametro `eval_metric` è stato impostato su `logloss`, metrica standard per i problemi di classificazione binaria, coerente con la funzione di perdita ottimizzata internamente dal modello.

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Insuccesso (0)	0.67	0.59	0.63	56
Successo (1)	0.56	0.64	0.60	45
Accuracy			0.61	

Tabella 5.2: Valutazione del modello XGBoost (una singola esecuzione)

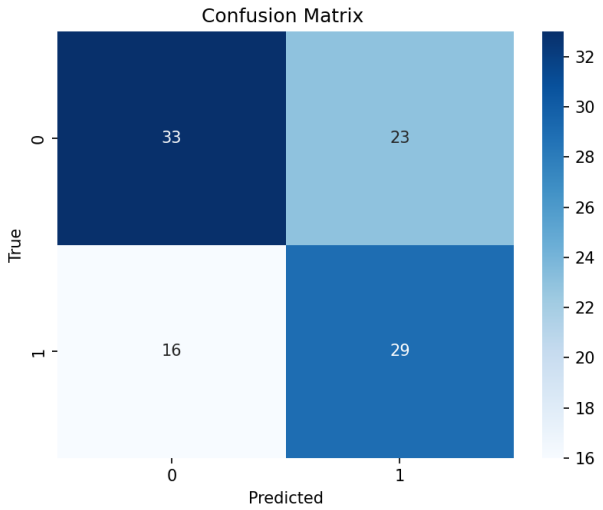

Figura 5.2: Matrice di confusione — XGBoost (una singola esecuzione)

K-Nearest Neighbors

La selezione degli iperparametri è stata effettuata tramite `RandomizedSearchCV`, con validazione stratificata 5-fold. I parametri esplorati sono stati:

- `knn__n_neighbors`: rappresenta il numero k di vicini da considerare. Sono stati testati valori da 3 a 40. Un valore basso rende il modello più sensibile al rumore, mentre un valore elevato fornisce maggiore stabilità ma può offuscare strutture locali.
- `knn__weights`: determina il peso assegnato a ciascun vicino durante la predizione. Le modalità valutate includono:
  - `uniform`: tutti i vicini contribuiscono in egual misura;
  - `distance`: i vicini più vicini contribuiscono maggiormente.
- `knn__metric`: specifica la metrica di distanza. Le metriche testate includono:
  - `euclidean` (distanza L2),
  - `manhattan` (distanza L1),
  - `chebyshev` (massima distanza lungo una singola dimensione).
- `knn__p`: parametro della distanza di Minkowski. `p=1` corrisponde alla distanza Manhattan, mentre `p=2` alla distanza Euclidea. È rilevante solo quando si utilizza la metrica `minkowski`.

Nel complesso, l'approccio basato su KNN si è dimostrato utile per esplorare strutture locali nei dati.

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Insuccesso (0)	0.69	0.59	0.63	56
Successo (1)	0.57	0.67	0.61	45
Accuracy		0.62		

Tabella 5.3: Valutazione del modello KNN (una singola esecuzione)

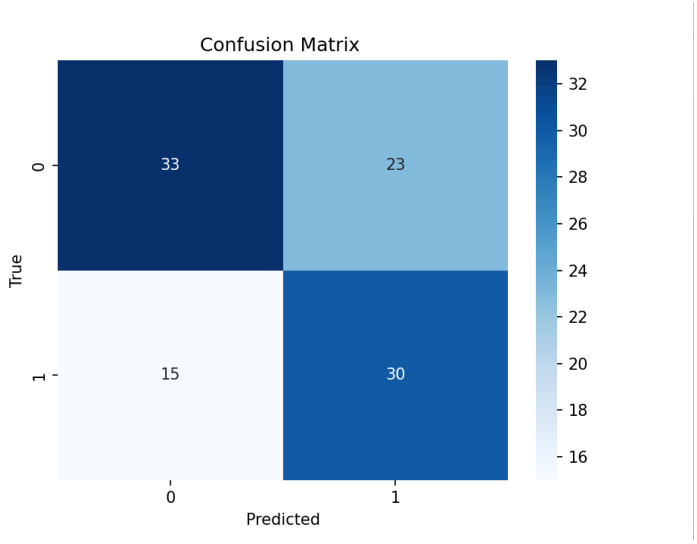

Figura 5.3: Matrice di confusione — KNN (una singola esecuzione)

Gradient Boosting Classifier

Il classificatore `GradientBoostingClassifier` di `scikit-learn` è stato utilizzato senza preprocessing tramite SMOTE o PCA. L'ottimizzazione degli iperparametri è stata effettuata con `RandomizedSearchCV` e validazione incrociata stratificata (5 fold). I principali iperparametri considerati sono:

- `n_estimators`: numero totale di alberi da costruire. Maggiore è il numero, maggiore la complessità del modello, ma anche il rischio di overfitting.
- `learning_rate`: coefficiente moltiplicativo che regola l'influenza di ciascun albero sulla predizione finale. Un valore basso (es. 0.01–0.1) migliora la generalizzazione, ma richiede più alberi.

- `max_depth`: profondità massima degli alberi base. Alberi più profondi catturano meglio interazioni complesse ma aumentano il rischio di overfitting.
- `subsample`: frazione del dataset usata per addestrare ciascun albero. Valori minori di 1 introducono casualità nel training (stochastic gradient boosting), utile per ridurre la varianza.
- `min_samples_split` e `min_samples_leaf`: controllano rispettivamente il numero minimo di campioni per dividere un nodo e per formare una foglia. Parametri elevati forniscono regolarizzazione e riducono l'overfitting.
- `max_features`: numero massimo di feature da considerare per ogni split. Valori come `'sqrt'` o `'log2'` possono migliorare la generalizzazione.

Questa architettura è particolarmente adatta in scenari in cui la relazione tra le feature e la variabile target è complessa e non lineare.

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Insuccesso (0)	0.65	0.70	0.67	56
Successo (1)	0.59	0.53	0.56	45
Accuracy		0.62		

Tabella 5.4: Valutazione del modello Gradient Boosting Classifier (una singola esecuzione)

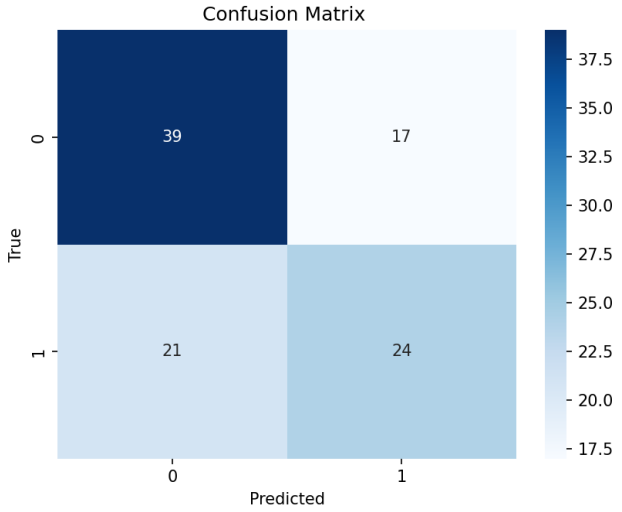

Figura 5.4: Matrice di confusione — Gradient Boosting Classifier (una singola esecuzione)

AdaBoost Classifier

La ricerca degli iperparametri è stata effettuata tramite `GridSearchCV` con validazione incrociata stratificata a 5 fold. I parametri ottimizzati sono:

- `clf__n_estimators`: numero di stimatori base (alberi). Valori testati: 50, 100, 300, 500, 1000. Un numero maggiore può migliorare la capacità predittiva, ma aumenta il rischio di overfitting e il costo computazionale.
- `clf__learning_rate`: fattore di riduzione del contributo di ogni albero aggiunto. Valori considerati: 0.001, 0.01, 0.05, 0.1, 0.5, 1.0. Un learning rate basso può migliorare la generalizzazione ma richiede un numero maggiore di stimatori.
- `clf__estimator__max_depth`: profondità massima degli alberi base. Valori: 1, 2, 3. Alberi poco profondi sono preferibili per ridurre l'overfitting, mantenendo la natura di classificatori deboli dell'algoritmo AdaBoost.

Questa combinazione di tecniche consente di affrontare efficacemente problemi di classificazione con classi sbilanciate, sfruttando il potere del boosting e la robustezza della normalizzazione e del bilanciamento dati.

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	Precision	Recall	F1-score	Support
Insuccesso (0)	0.76	0.62	0.69	56
Successo (1)	0.62	0.76	0.68	45
Accuracy			0.68	

Tabella 5.5: Valutazione del modello AdaBoost (una singola esecuzione)

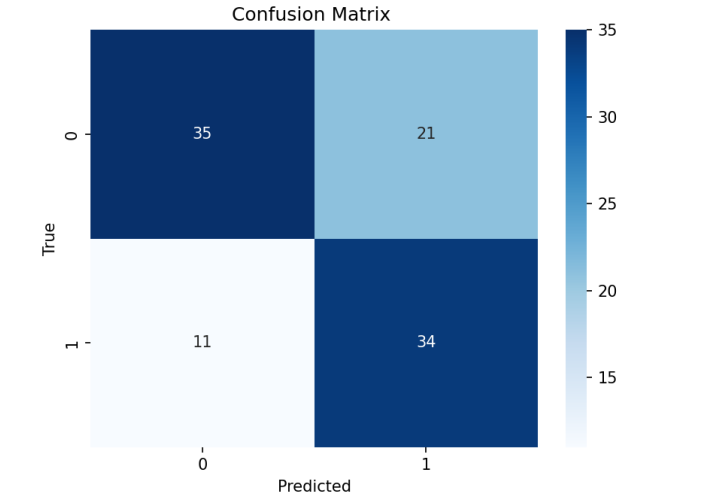

Figura 5.5: Matrice di confusione — AdaBoost (una singola esecuzione)

Support Vector Classifier (SVC)

La ricerca degli iperparametri è stata eseguita tramite `GridSearchCV` con validazione incrociata stratificata a 5 fold. I parametri ottimizzati sono:

- `clf__C`: parametro di regolarizzazione che controlla il compromesso tra massimizzazione del margine e minimizzazione dell'errore di classificazione. Valori testati: 0.1, 1, 10, 100. Valori più alti riducono la regolarizzazione, cercando di classificare correttamente tutti i punti di training.
- `clf__kernel`: funzione kernel usata per trasformare lo spazio delle feature. Sono stati testati i kernel lineare, RBF (Radial Basis Function) e polinomiale.
- `clf__gamma`: parametro del kernel RBF e polinomiale che definisce l'influenza di singoli esempi. Valori: `'scale'` (default, scala con varianza delle feature) e `'auto'` (inverso del numero di feature).

- `clf__degree`: grado del polinomio nel kernel polinomiale, testato per valori 2 e 3. Rilevante solo quando `kernel='poly'`.

L'utilizzo di SVC con questi parametri permette di modellare efficacemente sia problemi linearmente separabili sia situazioni più complesse con confini non lineari, con un controllo fine del bilanciamento bias-varianza tramite i parametri di regolarizzazione e kernel.

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Insuccesso (0)	0.70	0.84	0.76	56
Successo (1)	0.74	0.56	0.63	45
Accuracy		0.71		

Tabella 5.6: Valutazione del modello SVC (una singola esecuzione)

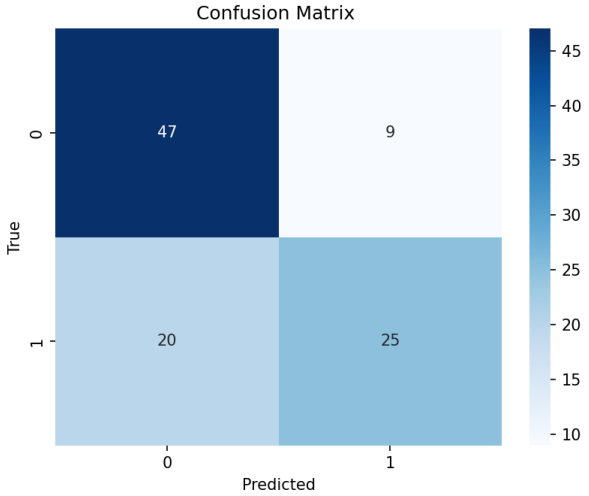

Figura 5.6: Matrice di confusione — SVC (una singola esecuzione)

5.3.4 Predizione aggregata

Tutti i modelli di classificazione descritti nelle sezioni precedenti hanno mostrato buone capacità predittive, sebbene con risultati soggetti a una certa variabilità tra le diverse esecuzioni. Per ridurre tale instabilità e ottenere una stima più robusta delle predizioni, è stata adottata una strategia di *ensemble learning* basata sul **voto di maggioranza**.

In particolare, una volta addestrati, tutti i modelli selezionati sono stati utilizzati congiuntamente per generare una singola predizione finale per ciascun nuovo progetto. Il meccanismo di aggregazione segue la logica riportata di seguito:

1. Ogni classificatore produce una predizione binaria (1 = successo, 0 = insuccesso);
2. Le predizioni dei modelli vengono sommate (`votes`);
3. Se la somma dei voti è superiore a 3 (ossia, più della metà dei sei modelli predice 1), il risultato aggregato è **1 (successo)**; altrimenti è **0 (insuccesso)**.

In caso di parità esatta (3 voti per ciascuna classe), viene assegnata la classe 0 (insuccesso), adottando un approccio conservativo.

5.3.5 Risultati e valutazione

La valutazione del modello ensemble è stata condotta tramite le metriche standard di `accuracy`, `precision`, `recall` e `f1-score`. Sebbene le prestazioni possano variare leggermente tra le diverse esecuzioni, l'accuratezza complessiva tende a collocarsi in un intervallo compreso tra il 64% e il 78%.

Per la stima finale è stato quindi impiegato un *ensemble classifier* basato sul meccanismo di majority voting: ciascun classificatore (Random Forest, XGBoost, KNN, Gradient Boosting, AdaBoost e SVC) ha generato le proprie predizioni sui dati di test, e la decisione finale per ogni istanza è stata determinata combinando le risposte secondo la regola sopra descritta.

La Tabella seguente riporta i risultati ottenuti in una singola esecuzione del sistema ensemble. Le due classi risultano complessivamente bilanciate in termini di prestazioni.

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Insuccesso (0)	0.74	0.71	0.73	56
Successo (1)	0.66	0.69	0.67	45
Accuracy		0.703		

Tabella 5.7: Valutazione dell'ensemble classifier tramite majority voting (una singola esecuzione)

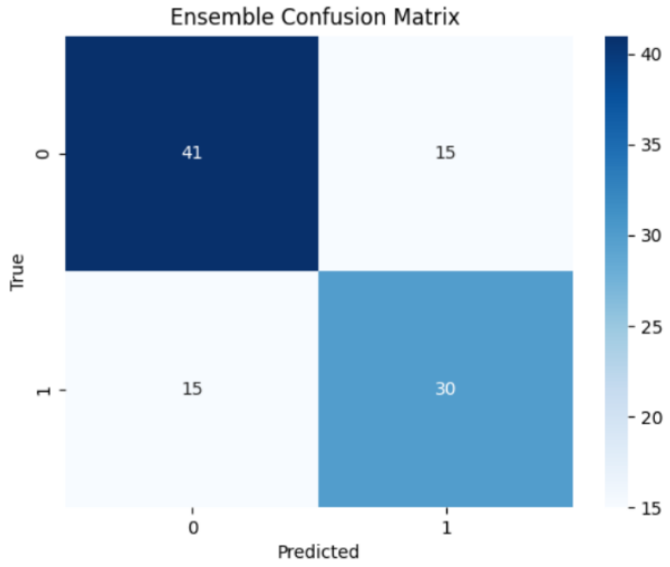

Figura 5.7: Matrice di confusione — Ensemble classifier (una singola esecuzione)

5.4 Classificazione Multiclasse

5.4.1 Definizione del target

Nella classificazione multiclasse, l'obiettivo è predire una classe qualitativa associata alla marginalità attesa del progetto. La variabile continua `marginality` è stata discretizzata in quartili utilizzando la funzione `qcut()` di Pandas, generando quattro classi ordinali:

- `Very bad`: quartile più basso di marginalità;
- `Not so good`;
- `Good`;
- `Very good`: quartile più alto di marginalità.

La marginalità era stata precedentemente calcolata attraverso la seguente formula:

$$M = \begin{cases} 1 - \dfrac{WD}{CE} & \text{se } WD \leq DE \\ -\left(\dfrac{WD}{CE} - 1\right) & \text{se } WD > DE \end{cases} \tag{5.2}$$

con:

- $M \in R$: marginalità economica del progetto;
- WD (`total_worked_days`): giorni effettivamente lavorati;
- CE (`commercial_estimates`): stima commerciale iniziale in giorni;
- $DE = 0{,}8 \cdot CE$ (`delivery_estimates`): stima di consegna operativa.

Nell'implementazione pratica, il rapporto $\frac{WD}{CE}$ viene troncato a 2 cifre decimali prima di essere utilizzato nel calcolo, e il risultato finale M viene anch'esso troncato a 2 decimali. In particolare, la distribuzione delle classi finale risulta composta in questo modo:

Classe	**Numero di istanze**
Very bad	81
Good	71
Very good	71
Not so good	45
Totale	268

Tabella 5.8: Distribuzione delle classi del target `marginality`

5.4.2 Suddivisione del training e test set

La suddivisione del dataset in training e test set per la classificazione multiclasse è stata realizzata mediante una funzione dedicata, riportata di seguito:

```
def prepare_data(df):
    X = df.drop(columns=['marginality'], errors='ignore')
    y = df['marginality']

    labels = ['Very bad', 'Not so good', 'Good', 'Very good']  # 4 classi
    y_class = pd.qcut(y, q=4, labels=labels)

    return train_test_split(X, y_class, test_size=0.2, shuffle=True)
```

La funzione `prepare_data()` riceve in input il *DataFrame* pre-processato e separa le variabili indipendenti (`X`) dalla variabile target (`y`), identificata come `marginality`. Poiché la marginalità è una variabile continua, essa viene discretizzata in quattro intervalli mediante la funzione `qcut()` della libreria `pandas`, che suddivide i valori in quartili con l'obiettivo di distribuire equamente le osservazioni tra le classi (circa il 25% per classe). Tuttavia, a causa della presenza di valori duplicati nella distribuzione della marginalità, la suddivisione non risulta perfettamente bilanciata, generando classi con un numero variabile di osservazioni. Le classi risultanti sono etichettate come *Very bad*, *Not so good*, *Good* e *Very good*, rappresentando rispettivamente livelli crescenti di marginalità.

Successivamente, il dataset viene suddiviso in due sottoinsiemi: il *training set* (80% dei dati) e il *test set* (20%). L'opzione `shuffle=True` consente di mescolare casualmente i campioni prima della suddivisione, riducendo eventuali bias legati all'ordine dei dati. Questa procedura garantisce una ripartizione bilanciata delle classi e consente di valutare in modo più affidabile le prestazioni dei modelli multiclasse.

5.4.3 Singoli Modelli

Per la classificazione multiclasse sono stati utilizzati i seguenti modelli: XGBoost Classifier, Random Forest Classifier e SVC. Il classificatore KNN è stato testato, ma poi escluso dalla predizione finale per prestazioni inferiori.

Di seguito si riportano l'analisi degli iperparametri selezionati per ciascun modello e la relativa valutazione delle prestazioni.

XGBoost Classifier

Nel contesto della classificazione multiclasse, XGBoost è stato addestrato utilizzando pesi di classe calcolati con `compute_class_weight`. L'ottimizzazione degli iperparametri è avvenuta con `GridSearchCV`, considerando:

- `n_estimators`: numero di alberi (100, 500, 1000);
- `learning_rate`: tasso di apprendimento, che regola l'impatto di ogni albero (0.01, 0.1, 0.2);
- `max_depth`: profondità massima degli alberi (3, 5, 7), per controllare la complessità.

L'algoritmo ha dimostrato buone capacità predittive, in particolare sulle classi più sbilanciate.

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Good	0.56	0.42	0.48	12
Not so good	0.36	0.21	0.27	19
Very bad	0.52	0.71	0.60	24
Very good	0.60	0.69	0.64	13
Accuracy		0.51		

Tabella 5.9: Valutazione del modello XGBoost Classifier (una singola esecuzione)

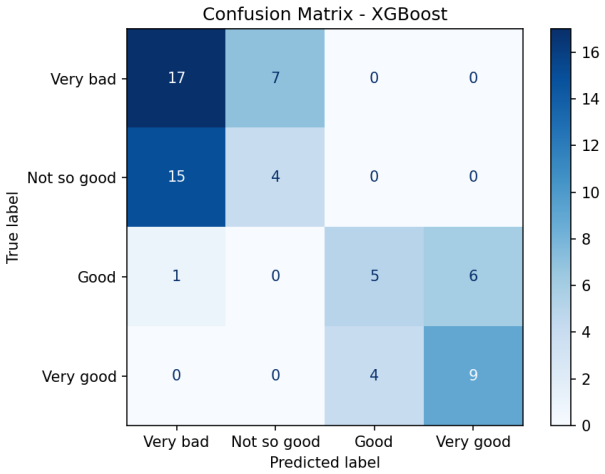

Figura 5.8: Matrice di confusione — XGBoost Classifier (una singola esecuzione)

Random Forest Classifier

Per la classificazione multiclasse, è stata utilizzata l'opzione `class_weight='balanced'` per gestire classi squilibrate.

Gli iperparametri esplorati tramite `GridSearchCV` sono:

- `n_estimators`: numero di alberi (100, 200, 300);
- `max_depth`: profondità massima degli alberi (10, 20, None);
- `min_samples_split`: minimo numero di campioni per suddividere un nodo (2, 5);
- `min_samples_leaf`: campioni minimi in una foglia (1, 2);
- `bootstrap`: uso del campionamento con rimpiazzo (True, False).

Il modello ha prodotto risultati bilanciati e accurati, con buone performance sulle classi centrali.

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	Precision	Recall	F1-score	Support
Good	0.75	0.25	0.38	12
Not so good	0.71	0.26	0.38	19
Very bad	0.59	0.92	0.72	24
Very good	0.60	0.92	0.73	13
Accuracy		0.62		

Tabella 5.10: Valutazione del modello Random Forest (una singola esecuzione)

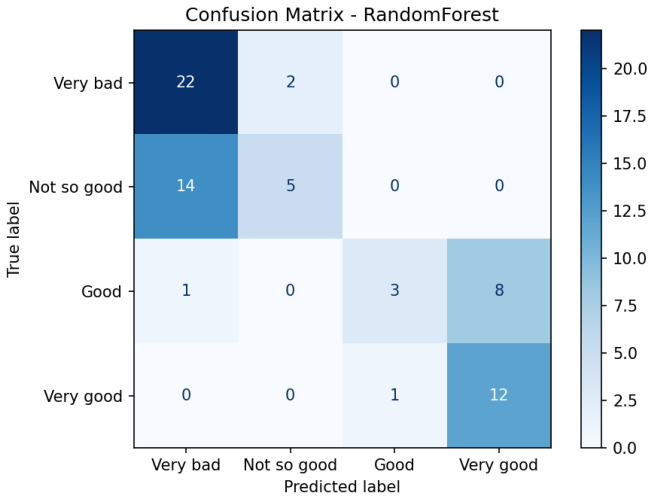

Figura 5.9: Matrice di confusione — Random Forest (una singola esecuzione)

Support Vector Classifier (SVC)

Gli iperparametri esplorati per il modello SVC sono:

- `C`: parametro di regolarizzazione (0.1, 1, 10, 100);
- `kernel`: funzione kernel (linear, rbf, poly);
- `gamma`: controllo dell'influenza di ogni punto (scale, auto);
- `degree`: grado del polinomio (2, 3) nel caso di `kernel='poly'`.

SVC ha mostrato stabilità nelle predizioni e una buona sensibilità nei confronti delle classi intermedie.

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	Precision	Recall	F1-score	Support
Good	0.42	0.42	0.42	12
Not so good	0.56	0.26	0.36	19
Very bad	0.56	0.79	0.66	24
Very good	0.69	0.69	0.69	13
Accuracy		0.56		

Tabella 5.11: Valutazione del modello SVC (una singola esecuzione)

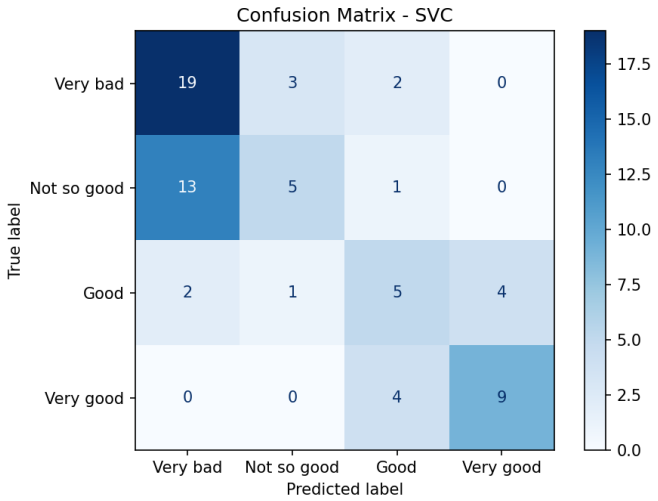

Figura 5.10: Matrice di confusione — SVC (una singola esecuzione)

K-Nearest Neighbors

Gli iperparametri esplorati per il modello K-Nearest Neighbors (KNN) sono:

- `n_neighbors`: numero di vicini considerati per la classificazione (1,2,3,4,5,10,12,15, 17,18,20,25,30,40,55,70);
- `weights`: schema di ponderazione dei vicini, che determina l'influenza dei punti più prossimi (`uniform, distance, inverse_distance`);
- `metric`: metrica utilizzata per il calcolo della distanza tra i punti (`cosine, euclidean, manhattan, chebyshev`).

Di seguito viene presentata la valutazione del modello, tenendo presente che i risultati ottenuti possono mostrare delle variazioni tra differenti esecuzioni.

Classe	Precision	Recall	F1-score	Support
Good	0.23	0.25	0.24	12
Not so good	1.00	0.05	0.10	19
Very bad	0.51	0.75	0.61	24
Very good	0.37	0.54	0.44	13
Accuracy		0.43		

Tabella 5.12: Valutazione del modello KNN (una singola esecuzione)

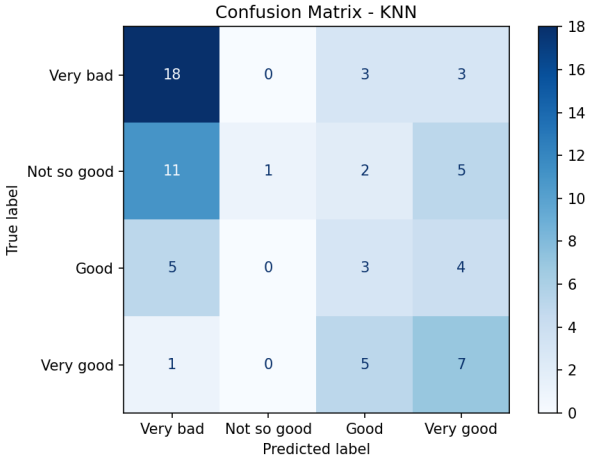

Figura 5.11: Matrice di confusione — KNN (una singola esecuzione)

Considerati i risultati meno soddisfacenti ottenuti da tale modello, si è scelto di escluderlo dalla fase di predizione aggregata.

5.4.4 Predizione aggregata

Per migliorare la robustezza delle predizioni, i tre modelli finali (XGBoost, Random Forest, SVC) sono stati combinati in un ensemble basato sulla media delle probabilità. Ogni modello ha restituito un vettore di probabilità per ciascuna classe, e la predizione finale è ottenuta scegliendo la classe con probabilità media massima.

Dato un insieme di modelli classificatori $M = \{M_1, M_2, M_3\}$, ciascun modello M_i restituisce per un'istanza x un vettore di probabilità $\mathbf{p}^{(i)} = [p_1^{(i)}, p_2^{(i)}, p_3^{(i)}]$, dove $p_j^{(i)}$ rappresenta la probabilità assegnata da M_i alla classe j-esima.

La probabilità aggregata per ciascuna classe j è ottenuta come:

$$\bar{p}_j = \frac{1}{|M|} \sum_{i=1}^{|M|} p_j^{(i)} \tag{5.3}$$

La classe predetta $\hat{y}$ è infine:

$$\hat{y} = \arg \max_{j \in \{1,2,3,4\}} \bar{p}_j \tag{5.4}$$

5.4.5 Risultati e valutazione

La valutazione è stata effettuata su un test set, utilizzando le metriche di `accuracy`, `precision`, `recall` e `f1-score`. Sebbene le prestazioni possano variare leg- germente tra le diverse esecuzioni, l'accuratezza complessiva tende a collocarsi in un intervallo compreso tra il 56% e il 70%. Nella tabella seguente sono riportati i risultati ottenuti in una singola esecuzione:

Classe	**Precision**	**Recall**	**F1-score**	**Support**
Good	0.55	0.43	0.48	14
Not so good	0.71	0.45	0.56	11
Very bad	0.79	0.90	0.84	29
Very good	0.59	0.71	0.65	14
Accuracy		0.69		

Tabella 5.13: Valutazione dell'ensemble multiclasse (una singola esecuzione)

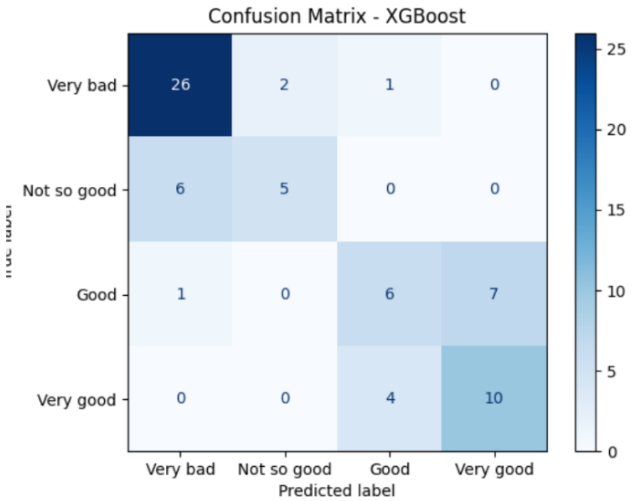

Figura 5.12: Matrice di confusione — Classificazione multiclasse (una singola esecuzione)

Capitolo 6

Ricerca Semantica con VectorDB

6.1 Obiettivo e motivazione

Nel contesto del sistema di supporto alla team composition, è stato integrato un meccanismo di **ricerca semantica** basato su VectorDB per recuperare progetti simili a uno nuovo in fase di proposta. Lo scopo è fornire un ulteriore supporto decisionale, consentendo di confrontare il progetto in esame con progetti storici simili in termini semantici, e osservare i relativi esiti (marginalità, successo, team impiegato).

Un elemento distintivo dell'applicativo finale consiste nell'utilizzo diretto dei progetti più simili per supportare la **composizione automatica del team**: i membri del team associato al progetto con embedding più vicino vengono proposti come base per il nuovo progetto, rendendo il sistema non solo predittivo, ma anche generativo rispetto alla composizione progettuale. Questo approccio si basa sull'assunzione che progetti semanticamente simili presentino caratteristiche organizzative compatibili, e sfrutta in modo efficace la conoscenza implicita contenuta nello storico.

Il sistema di ricerca è stato costruito utilizzando **Milvus**, una soluzione specializzata per la gestione di database vettoriali su larga scala [14], progettata per supportare ricerche approssimate di similarità con efficienza e scalabilità. La ricerca semantica consente di superare i limiti della ricerca tradizionale basata su keyword o matching esatto, offrendo risultati più pertinenti anche in presenza di variazioni lessicali o strutturali nelle descrizioni.

6.2 Architettura e tecnologia utilizzata

La soluzione implementata sfrutta:

- **Milvus** , un sistema di vector database ad alte prestazioni progettato per archiviare e cercare efficientemente vettori ad alta dimensione;

- **Sentence Transformers**, una libreria basata su BERT per la generazione di embedding semantici a partire da testi in linguaggio naturale;
- **pymilvus**, client Python per interagire con Milvus;
- **modello paraphrase-MiniLM-L3-v2**, un modello di Sentence-BERT preaddestrato adatto alla generazione di embedding compatti (384 dimensioni).

Ogni progetto viene rappresentato da un embedding numerico generato a partire da una descrizione sintetica contenente le informazioni chiave del progetto (procedura, stima commerciale, livello di rischio). I vettori risultanti sono indicizzati in Milvus per effettuare ricerche per similarità (similarità coseno o distanza euclidea normalizzata).

6.3 Generazione delle descrizioni ed embedding

Per ogni progetto storico, viene costruita una descrizione testuale nella forma:

``‘procedure: *X*, stima_commerciale: *Y*, livello_di_rischio: *Z*’’

Questa frase sintetizza in linguaggio naturale le caratteristiche salienti del progetto. Il campo `procedure` è estratto dal codice progetto (rimuovendo eventuali numeri), mentre `stima_commerciale` e `livello_di_rischio` sono attributi originari.

La descrizione viene poi embeddizzata tramite il modello `paraphrase-MiniLM-L3-v2`:

6.4 Struttura della collezione in Milvus

La collezione principale, denominata `demo_collection`, contiene campi strutturati (numerici e categoriali) e un campo `vector` di tipo `FLOAT_VECTOR` con dimensione 384. Ogni entry rappresenta un progetto e include, oltre al vettore, metadati rilevanti:

- codice progetto (`codice_epic`),
- assegnatario, customer, tecnologie, membri,
- success/unsuccess, marginalità,
- descrizione testuale (`description`),
- vettore semantico (`vector`).

L'inserimento nella collezione avviene tramite API `insert` di `MilvusClient`, dopo creazione dello schema e verifica di validità.

6.5 Ricerca semantica

La ricerca di progetti simili avviene come segue:

1. Viene generata una descrizione testuale del nuovo progetto secondo la stessa logica (procedure, stima commerciale, livello di rischio);
2. Tale descrizione viene trasformata in un vettore tramite SentenceTransformer;
3. Viene effettuata una ricerca approssimata nel database vettoriale Milvus, utilizzando la funzione `search()`;
4. I risultati vengono filtrati sulla base di una soglia di distanza (`distance_threshold`), calcolata mediante la metrica di **Cosine Similarity**, per garantire pertinenza semantica;
5. Per ogni progetto simile trovato, vengono restituite tutte le informazioni rilevanti (membri, tecnologie, marginalità, outcome).

Il codice di ricerca utilizza la seguente logica:

```
results = client.search(
    collection_name='demo_collection',
    data=query_vector,
    limit=3,
    output_fields=[...]
)
filtered = [r for r in results[0] if r['distance'] >= 0.6]
```

La soglia di similarità (`0.6`) è stata scelta empiricamente per bilanciare precisione e varietà nei risultati.

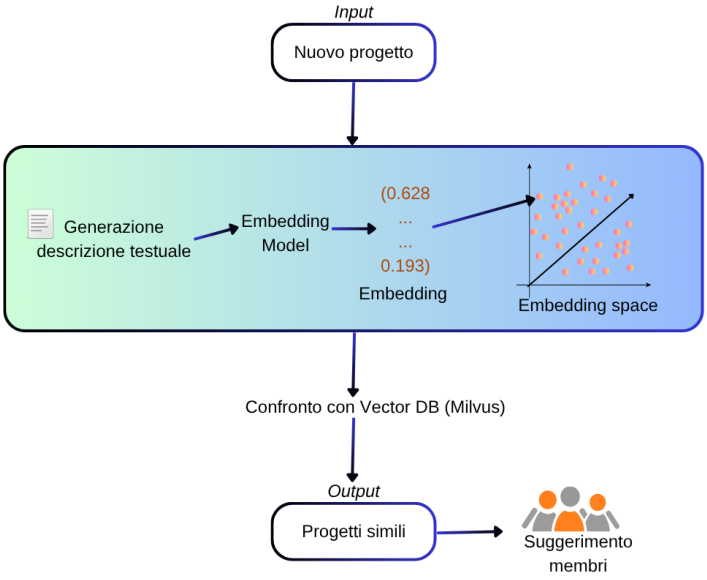

Figura 6.1: Schema illustrativo del funzionamento della ricerca semantica con VectorDB

6.6 Output e utilizzo applicativo

`L'output` della ricerca viene salvato in formato JSON ed utilizzato all'interno del sistema per mostrare all'utente i progetti storici più simili. Per ogni progetto simile, vengono mostrate:

- Informazioni strutturate (cliente, team, tecnologie, outcome),
- La descrizione testuale,
- La distanza semantica rispetto al progetto in analisi.

La ricerca semantica basata su VectorDB ha rappresentato un importante complemento ai modelli predittivi sviluppati nella presente tesi. In particolare, l'integrazione di `Milvus` come motore vettoriale ha consentito di arricchire il sistema decisionale con una componente di similarità semantica tra progetti, utile per individuare esperienze pregresse con caratteristiche affini al nuovo progetto in fase di pianificazione.

Nell'applicativo finale, i team suggeriti per i nuovi progetti non vengono generati ex novo, ma derivano dalla composizione dei team impiegati nei progetti più simili individuati dal motore vettoriale. Questa strategia sfrutta la conoscenza storica e promuove la riutilizzabilità delle configurazioni organizzative che si sono dimostrate efficaci in contesti analoghi.

Capitolo 7

Analisi della Skill Matrix

La **Skill Matrix** è un documento, fornito in formato Excel, che contiene informazioni strutturate sulle competenze dei dipendenti aziendali, suddivise per tecnologia, settore di appartenenza e ruolo. In questo lavoro, la matrice viene utilizzata per selezionare automaticamente i candidati più adatti a partecipare ad un nuovo progetto, sulla base delle tecnologie richieste e del settore di appartenenza del cliente.

7.1 Formato e struttura della Skill Matrix

Ciascun file Excel analizzato contiene una tabella in cui ogni riga rappresenta un dipendente, e le colonne forniscono:

- `usr_name`: nome del dipendente;
- `role`: ruolo ricoperto;
- `industry`: settore aziendale di riferimento;
- `Tecnologie`: ciascuna tecnologia ha una colonna dedicata, contenente il livello di competenza della risorsa.

I livelli di competenza sono codificati come segue:

Valore	Descrizione
0	Non applicabile
1	Livello Base: conoscenza elementare, non autonoma
2	Livello Avanzato: attività completate in autonomia con buona qualità
3	Livello Esperto: autonomia + capacità di formare altri
4	Livello Specialista: autonomia, ottimizzazione e implementazione soluzioni

Tabella 7.1: Legenda dei livelli di competenza nella Skill Matrix

7.2 Estrazione e pre-elaborazione dei dati

Durante il caricamento del file Excel nell'applicazione, questo viene convertito in formato JSON tramite un agente automatico. I dati vengono unificati in una lista di dizionari Python, normalizzando i nomi delle tecnologie e filtrando eventuali duplicati.

Successivamente, viene applicato un primo filtro basato sul settore (`industry`) del cliente del progetto.

7.3 Selezione dei dipendenti più esperti

Una volta filtrati i dipendenti per settore, si procede con la selezione delle tecnologie comuni tra quelle richieste dal nuovo progetto e quelle presenti nella skill matrix. Queste tecnologie sono poi utilizzate per determinare la rilevanza di ciascun dipendente.

Un dipendente viene considerato idoneo se possiede un livello di competenza almeno pari a 2 (cioè "Livello Avanzato") in **tutte** le tecnologie comuni richieste.

Formalmente, dato un insieme di tecnologie comuni $T = \{t_1, t_2, ..., t_k\}$ e un dipendente d_i con livelli $l_i(t_j)$, d_i viene selezionato se:

$$\forall t_j \in T, \quad l_i(t_j) \geq 2$$

Questa soglia consente di selezionare solo risorse in grado di completare le attività assegnate in autonomia, garantendo al contempo un buon livello di competenza.

7.4 Output e utilizzo applicativo

Al termine del processo, l'applicazione restituisce un elenco interattivo dei dipendenti selezionati, comprensivo di:

- Nome, ruolo e settore;
- Tecnologie possedute con livello ≥ 2;

La Skill Matrix rappresenta un elemento importante del sistema di supporto alla composizione del team, in quanto consente di integrare informazioni aggiornate e strutturate sulle competenze tecniche dei dipendenti.

Dopo l'individuazione automatica dei progetti simili tramite ricerca semantica, l'applicazione propone delle ipotesi di team composte dai membri storicamente coinvolti nei progetti più affini. Questa proposte iniziali possono essere integrate o modificate attraverso l'analisi della matrice delle competenze.

In particolare, il sistema filtra automaticamente i profili dei dipendenti selezionando quelli con un livello di competenza pari o superiore a "Avanzato" (livello 2) nelle tecnologie richieste dal nuovo progetto. Questo consente all'utente di aggiungere ai team ulteriori risorse qualificate, mantenendo un elevato grado di flessibilità e controllo umano.

L'integrazione tra memoria storica (via progetti simili) e competenza attuale (via Skill Matrix) rende la composizione finale del team bilanciata, coerente con il contesto operativo e orientata al successo progettuale.

Capitolo 8

Interfaccia Utente e Deployment

8.1 Interfaccia utente

Dopo aver completato l'implementazione dei singoli componenti logici del sistema, è stato necessario integrare tutte le funzionalità in un unico flusso applicativo coerente. Il risultato finale è un'applicazione web interattiva, che guida l'utente attraverso una sequenza strutturata di operazioni, ciascuna finalizzata alla composizione di un team ottimale per un nuovo progetto aziendale.

Per lo sviluppo dell'interfaccia grafica è stata utilizzata la libreria **Gradio**, un framework open source in Python che permette di realizzare rapidamente interfacce web . Gradio consente di definire funzioni Python e associarle a componenti visuali interattivi, come bottoni, campi di upload, slider o tabelle, generando automaticamente l'interfaccia corrispondente.

L'applicativo finale è organizzato in cinque step sequenziali, ciascuno rappresentante una fase logica del processo. Ogni fase è accessibile all'utente tramite un layout chiaro e guidato, con la possibilità di visualizzare o modificare i dati a ogni passaggio. Di seguito viene descritto il funzionamento di ciascuna sezione.

1. Datasets In questa fase iniziale, l'utente può caricare o sostituire i file di input. I principali dataset richiesti includono:

- **Storico progetti**: contiene i dati dei progetti passati, utilizzati per addestrare i modelli di machine learning.
- **Organigramma**: include le informazioni sui dipendenti attivi, specificando ruolo e settore di appartenenza (es. Banking, Fashion).

Questi file costituiscono la base informativa su cui si fonda l'intero sistema e condizionano le analisi e le predizioni successive.

2. Project Charter L'utente può caricare un documento `.pdf` contenente il *project charter* del nuovo progetto da avviare. Questo documento, che non include ancora un team, viene elaborato automaticamente tramite un agente AI che ne estrae le informazioni salienti (es. titolo, obiettivi, tecnologie richieste). I dati estratti vengono visualizzati a schermo per consentire una rapida verifica da parte dell'utente.

3. Similar Projects In questa fase, l'applicativo interroga il database aziendale per individuare i progetti storici più simili al nuovo progetto caricato. La similarità si basa sulle caratteristiche descritte nel capitolo dedicato. I team utilizzati nei progetti simili vengono proposti come possibili configurazioni iniziali per il nuovo progetto. Questo approccio consente di partire da soluzioni già collaudate in passato.

4. Skill Matrix All'utente viene richiesto di caricare un file `.xlsx` contenente la skill matrix aziendale, ovvero una matrice che associa a ogni dipendente il livello di competenza (da 0 a 4) in una serie di tecnologie e strumenti. L'applicativo analizza la matrice per identificare i profili più adatti alle tecnologie richieste dal nuovo progetto. Nella fase finale l'utente avrà la possibilità di modificare manualmente le proposte di team, aggiungendo o rimuovendo membri suggeriti dalla Skill Matrix.

5. Prediction Infine, vengono applicati i modelli di machine learning addestrati per stimare la probabilità di successo di ciascuna proposta di team. In particolare, vengono utilizzate due tipologie di classificazione:

- **Classificazione binaria**: restituisce un esito di successo (`1`) o insuccesso (`0`) del progetto.
- **Classificazione multiclasse**: assegna un livello qualitativo al potenziale successo: `Very Bad`, `Not So Good`, `Good`, `Very Good`.

I risultati ottenuti, visualizzati in modo interattivo, forniscono all'utente informazioni utili per scegliere in modo consapevole la configurazione finale del team. All'utente viene anche data la possibilità di modificare i team proposti dall'applicativo aggiungendo o rimuovendo i membri consigliati dall'analisi della Skill Matrix.

Di seguito immagini esemplificative del funzionamento dell'applicativo finale, i dati personali e sensibili sono stati censurati per privacy.

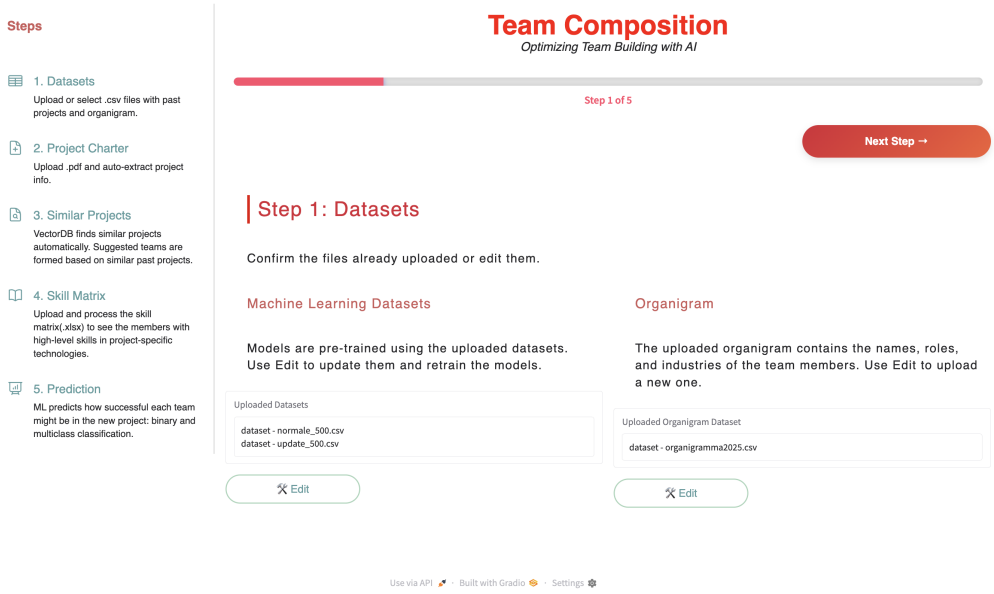

Figura 8.1: Schermata del primo step

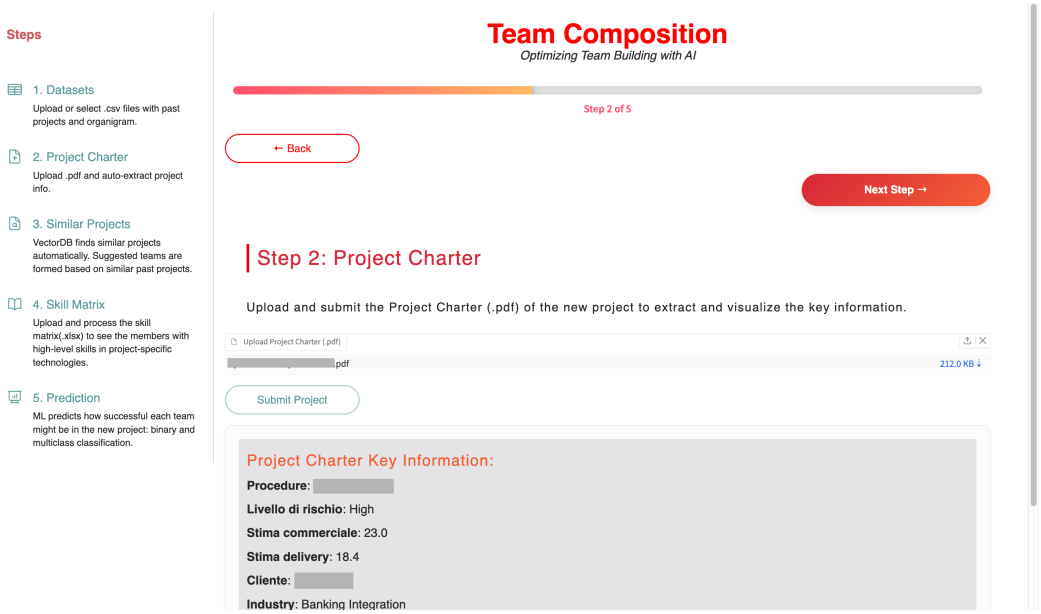

Figura 8.2: Schermata del secondo step

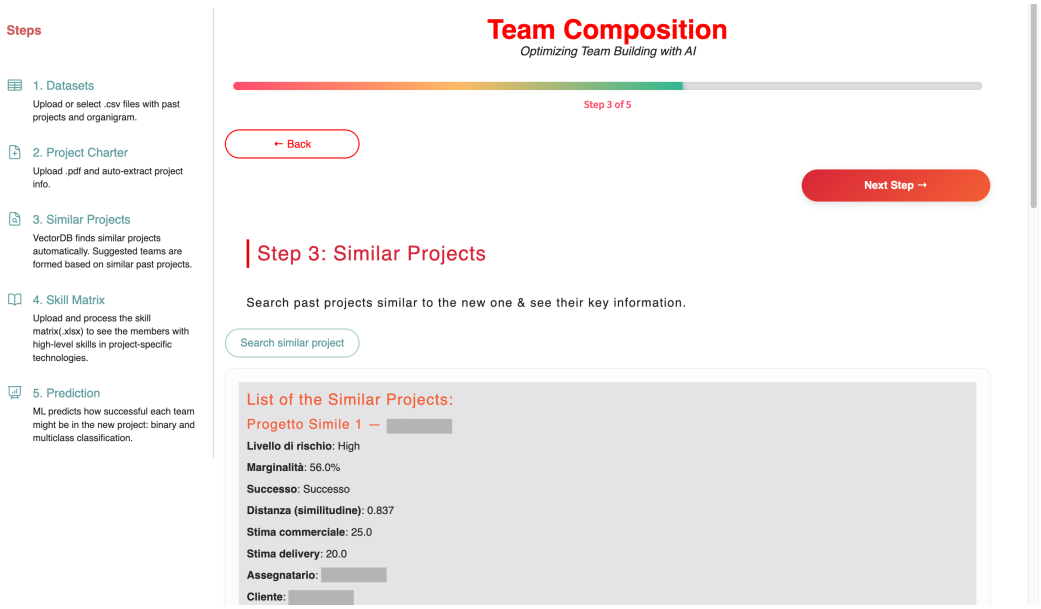

Figura 8.3: Schermata del terzo step

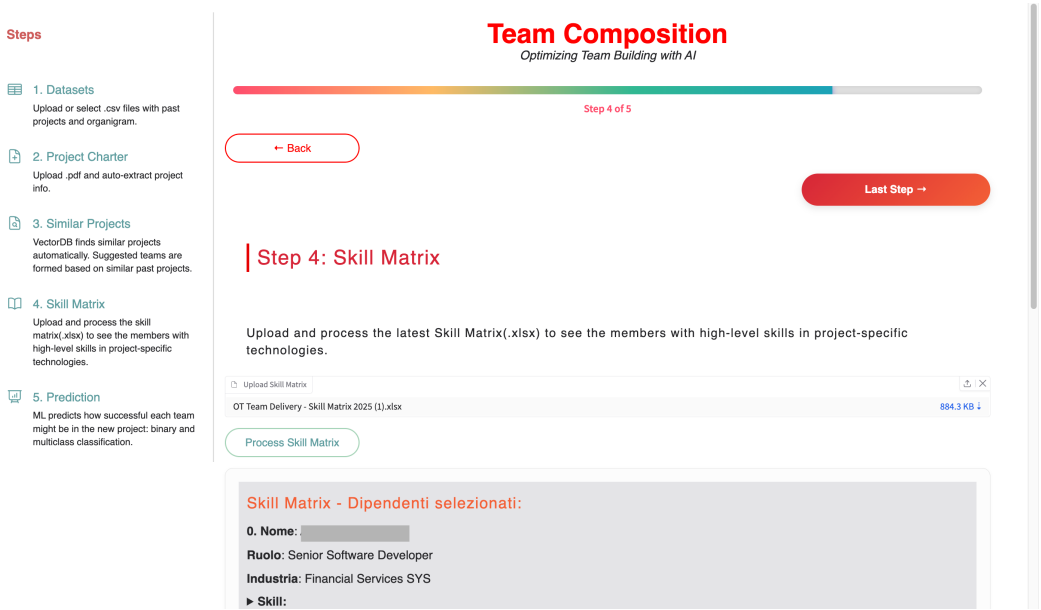

Figura 8.4: Schermata del quarto step

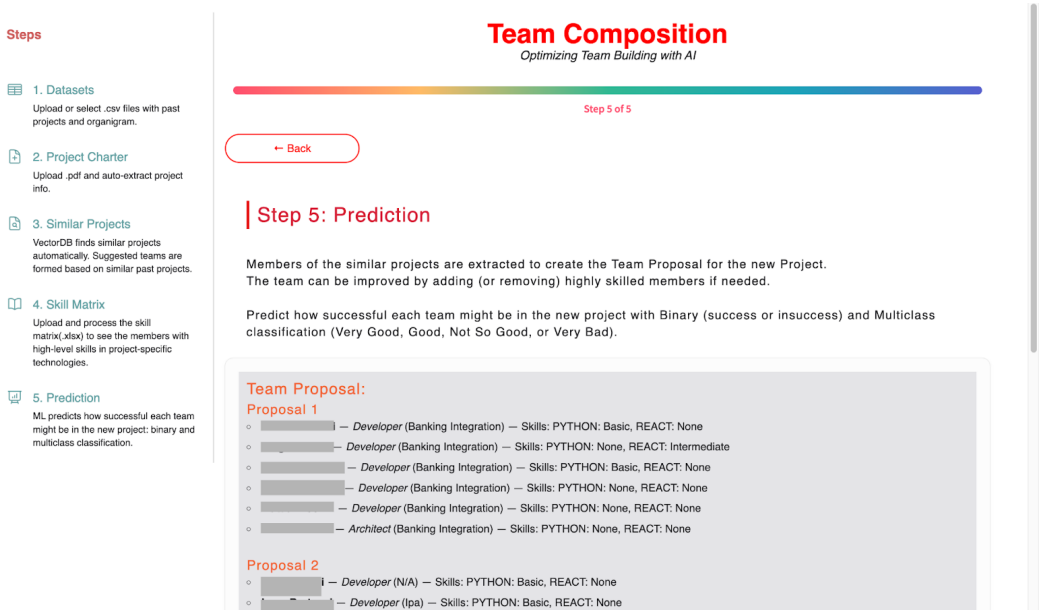

Figura 8.5: Schermata del quinto step - prima parte

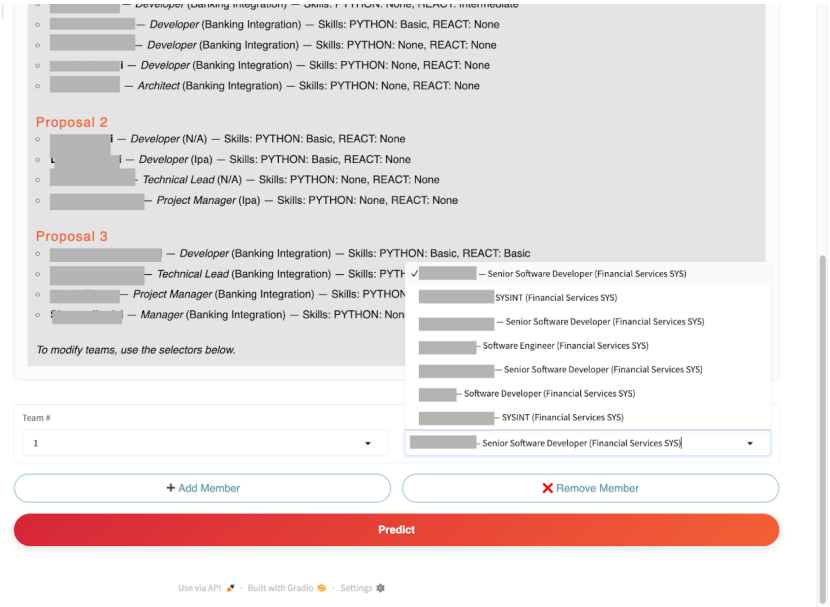

Figura 8.6: Schermata del quinto step - seconda parte

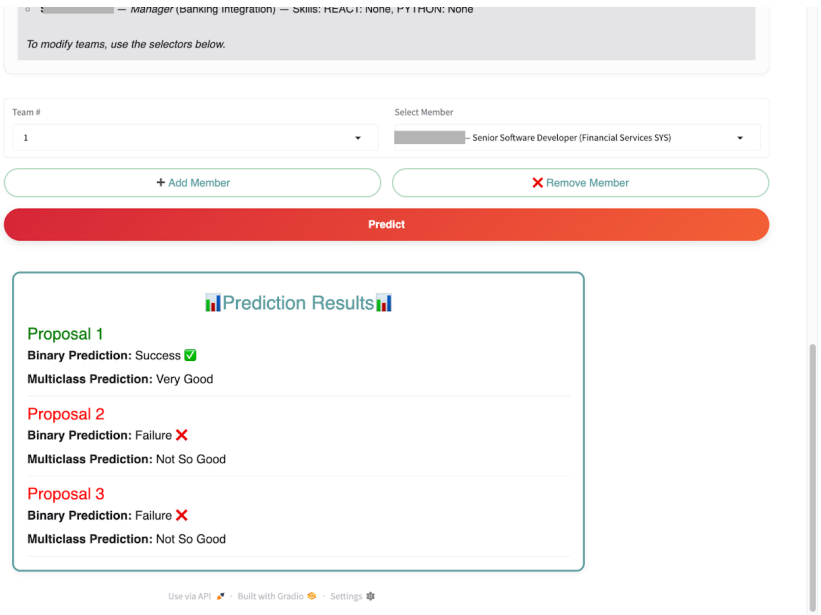

Figura 8.7: Schermata del quinto step - terza parte, le previsioni ottenute sono solo esemplificative

8.2 Deployment

Parallelamente allo sviluppo dell'interfaccia, è stato implementato un modulo separato per l'esposizione dei modelli tramite API REST, utilizzando il framework **FastAPI**. Tale modulo consente il caricamento e la messa a disposizione dei modelli addestrati attraverso endpoint dedicati, in grado di ricevere dati in input e restituire predizioni in formato JSON.

Questo sistema di deployment è stato progettato come componente *standalone* e non è attualmente integrato nell'interfaccia realizzata con Gradio.

Per pubblicare il servizio è stata utilizzata la piattaforma **Render**, che consente di effettuare il deploy automatico di applicazioni web o API a partire da repository Git. Render gestisce l'esecuzione del server FastAPI, l'assegnazione di un endpoint pubblico e il ciclo di aggiornamento continuo a ogni modifica del codice sorgente.

Il deployment con FastAPI su Render include:

- Caricamento dei modelli salvati (tramite `joblib`).
- Definizione di endpoint REST per la classificazione binaria e multiclasse.
- Validazione dei dati in input e gestione degli errori.

- Risposte in formato JSON contenenti le probabilità associate a ciascuna classe e la previsione finale.

Pur non essendo utilizzato attivamente nell'applicativo finale, questo modulo rappresenta una base solida per eventuali estensioni future che prevedano il disaccoppiamento tra frontend e backend.

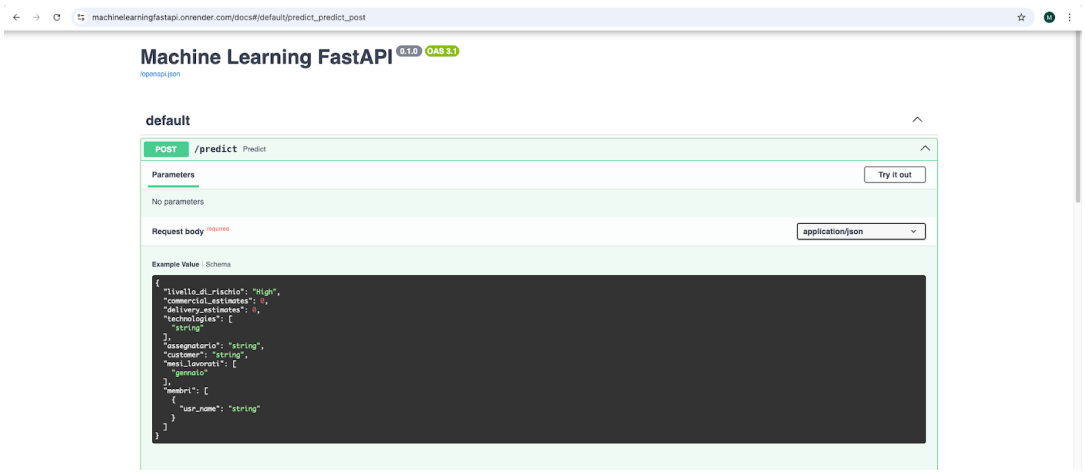

Figura 8.8: Input richiesto dal metodo POST della fastApi dei modelli di ML

Capitolo 9

Approcci Alternativi e Prospettive di Sviluppo Futuro

Il sistema sviluppato nella presente tesi si basa su algoritmi di machine learning classici per la predizione del successo progettuale e su tecniche di ricerca semantica tramite VectorDB per l'identificazione di progetti simili. Sebbene l'approccio adottato abbia dimostrato risultati promettenti, esistono diverse metodologie alternative che potrebbero essere esplorate per migliorare le prestazioni del sistema.

Questo capitolo presenta quattro approcci complementari: le reti neurali, la programmazione a vincoli, l'ottimizzazione combinatoria e i problemi di assegnazione.

9.1 Reti Neurali per la Predizione della Performance di Team

Le reti neurali artificiali (ANN) rappresentano una classe di modelli di machine learning capaci di apprendere relazioni complesse tra le variabili in modo automatico. A differenza dei modelli classici utilizzati in questo lavoro, le reti neurali possono scoprire autonomamente pattern nascosti nei dati senza richiedere un'elaborazione manuale delle caratteristiche.

[2] Uno studio condotto presso il Project Management Institute ha dimostrato che l'utilizzo di modelli ANN permette di predire il 74,3% della performance dei team. La ricerca ha evidenziato come le reti neurali siano in grado di catturare relazioni derivanti dalle performance individuali passate e dalle loro interazioni, risultando particolarmente efficaci quando i dati presentano interdipendenze complesse.

Un'estensione interessante è rappresentata dalle **Graph Neural Networks (GNNs)**, che permettono di rappresentare i dipendenti come nodi di un grafo e le loro collaborazioni passate come collegamenti tra nodi. [4] Il framework MENTOR utilizza questa tecnica per modellare tre aspetti fondamentali: la struttura delle relazioni nel team (aspetto

topologico), l'importanza di ciascun membro (aspetto di centralità) e il contesto organizzativo (aspetto contestuale). Questo approccio permette al modello di apprendere quali configurazioni di collaborazione sono più efficaci.

9.2 Constraint Programming per la Composizione di Team

La programmazione a vincoli (Constraint Programming, CP) è un approccio che permette di specificare esplicitamente le condizioni che una soluzione deve soddisfare, lasciando poi al sistema il compito di trovare combinazioni valide. A differenza dei modelli predittivi che suggeriscono soluzioni senza garantirne la fattibilità, il CP assicura che tutte le proposte rispettino i vincoli definiti.

[3] IBM Global Services ha sviluppato un sistema basato su constraint programming per assegnare lavoratori altamente qualificati a progetti complessi. Il sistema gestisce centinaia di posizioni e risorse considerando vincoli come: competenze richieste, posizione geografica, lingua, disponibilità temporale e possibilità di formazione. Il vantaggio principale è che tutte le soluzioni proposte sono garantite essere realizzabili.

In un sistema CP, si definiscono:

- Le **variabili**: ad esempio, quale dipendente assegnare a quale progetto
- I **domini**: i valori possibili per ciascuna variabile (quali dipendenti sono disponibili)
- I **vincoli**: le regole da rispettare (ad esempio: "ogni progetto deve avere almeno un membro con competenza Java di livello 3 o superiore", "nessun dipendente può lavorare su più di 3 progetti contemporaneamente")

Il sistema esplora automaticamente le possibili combinazioni, scartando quelle che violano i vincoli, fino a trovare soluzioni ammissibili. Il constraint programming potrebbe essere integrato come componente di validazione nel sistema esistente. Il flusso di lavoro diventerebbe:

9.3 Ottimizzazione Combinatoria per la Selezione Ottimale

L'ottimizzazione combinatoria affronta il problema di trovare la migliore soluzione possibile tra un numero finito (ma spesso molto grande) di alternative. Nel contesto della composizione di team, l'obiettivo è selezionare il gruppo di persone che massimizza la probabilità di successo del progetto, rispettando vincoli come disponibilità, competenze e budget.

[1] Il Multiple Team Formation Problem (MTFP) è un esempio di problema di ottimizzazione combinatoria applicato alla formazione di team. Permette di gestire situazioni complesse dove:

- Servono team multipli per progetti diversi
- Ogni progetto richiede competenze specifiche in quantità diverse
- Ciascun dipendente può dedicare frazioni del proprio tempo a progetti diversi
- Esistono vincoli sul numero minimo e massimo di membri per team

Il problema viene formulato matematicamente specificando cosa si vuole massimizzare (ad esempio, la somma delle probabilità di successo di tutti i progetti) e quali vincoli devono essere rispettati. Strumenti specializzati (solver) esplorano lo spazio delle soluzioni per trovare quella ottimale.

9.4 Problemi di Assegnazione e Algoritmo Ungherese

Il problema di assegnazione è un caso particolare di ottimizzazione dove si deve stabilire una corrispondenza uno-a-uno tra due insiemi: ad esempio, assegnare dipendenti a ruoli o progetti a team leader. L'obiettivo è minimizzare il costo totale (o massimizzare il beneficio totale) di tutte le assegnazioni considerate insieme.

[12] L'**Algoritmo Ungherese**, sviluppato da Harold Kuhn nel 1955, risolve questo problema in modo efficiente. L'algoritmo lavora su una matrice di costi dove ogni cella indica quanto 'costa' assegnare una persona a un determinato compito. Attraverso una serie di trasformazioni matematiche della matrice, l'algoritmo identifica l'assegnazione ottimale che minimizza il costo totale.

L'algoritmo è particolarmente efficiente: anche per problemi di dimensioni medie (ad esempio, 100 persone da assegnare a 100 compiti) trova la soluzione ottimale in tempi rapidi. L'algoritmo ungherese sarebbe utile in scenari dove l'azienda deve prendere decisioni di assegnazione su scala più ampia. Ad esempio:

Una limitazione importante è che l'algoritmo richiede un numero uguale di persone e compiti. Se ci sono più persone che ruoli (o viceversa), è necessario aggiungere elementi 'fittizi' per bilanciare la matrice.

L'esplorazione di questi approcci alternativi rappresenta uno spunto per ricerche future, con il potenziale di trasformare il sistema da strumento di supporto decisionale a sistema di ottimizzazione automatica della composizione dei team aziendali.

Capitolo 10

Conclusioni

L'obiettivo principale di questa tesi è stato lo sviluppo di un sistema intelligente a supporto della composizione di team aziendali, basato su dati reali e tecniche di apprendimento automatico. Il lavoro svolto ha portato alla realizzazione di un'applicazione web interattiva, strutturata in più fasi consecutive, ognuna delle quali contribuisce alla definizione della migliore configurazione possibile per affrontare un nuovo progetto.

L'intero processo è stato pensato per offrire all'utente un'esperienza guidata: dalla selezione dei dati di partenza, all'estrazione automatica delle informazioni contenute nel project charter, fino alla ricerca dei progetti più simili nel database storico, alla selezione dei membri più competenti tramite skill matrix e, infine, alla predizione del potenziale successo delle varie proposte di team. Questo approccio modulare ha permesso di integrare differenti componenti tecniche e logiche, in un sistema coerente e utilizzabile anche da figure non tecniche.

Il sistema si dimostra utile non solo nella fase di composizione del team, ma anche come strumento di analisi preventiva del progetto stesso, offrendo un primo livello di valutazione che mira a prevedere l'esito progettuale attraverso la classificazione predittiva. L'adozione congiunta di classificazione binaria e multiclasse consente di ottenere sia una valutazione netta (successo/insuccesso), sia un'indicazione qualitativa più sfumata sull'esito atteso.

Durante lo sviluppo sono emerse alcune *criticità* significative. La più rilevante riguarda la qualità dei dati aziendali storici: numerosi valori risultavano mancanti, parziali o non aggiornati. Questo ha reso necessario un intenso lavoro di pulizia e bilanciamento, con inevitabili compromessi. Tuttavia, è ragionevole supporre che un'organizzazione futura più rigorosa nella raccolta e gestione dei dati, ad esempio tramite sistemi gestionali integrati o strumenti similari, possa aumentare notevolmente la precisione dei modelli predittivi.

Un'ulteriore osservazione riguarda la sensibilità dei modelli rispetto alla composizione del team: in alcuni casi, anche modificando i membri proposti, la previsione finale restava invariata. Questo indica che non sempre la composizione del team è un fattore dominante

nel determinare l'esito previsto, o che i dati disponibili non catturano in modo sufficiente le variabili rilevanti. Ciononostante, il sistema oltre a fornire una predizione dell'esito progettuale, anche se non sempre legata al team, si rivela comunque uno strumento di supporto valido per supportare la creazione di un team, fornendo indicazioni oggettive su cui basare decisioni strategiche.

In prospettiva futura, il sistema potrà essere esteso e potenziato in diverse direzioni come esposto nel capitolo dedicato.

Nel complesso, l'applicativo sviluppato rappresenta un esempio concreto di come i dati e l'intelligenza artificiale possano essere utilizzati per supportare scelte complesse nel contesto aziendale, con benefici potenziali in termini di efficienza, qualità delle decisioni e gestione strategica delle risorse umane.

Appendice A
Prompt per l'estrazione automatica dei dati dal Project Charter

Prompt per l'estrazione automatica dei dati dal Project Charter

Nel presente allegato si riporta il prompt testuale utilizzato dal componente di estrazione automatica dei dati (*Data Extraction Agent*) per l'analisi dei *project charter*. Il prompt è stato progettato per guidare il modello linguistico (GPT) nella comprensione del documento in formato PDF e nella restituzione delle informazioni chiave in formato strutturato.

Il testo seguente viene dinamicamente compilato e fornito come input al modello, insieme al contenuto testuale estratto dal PDF:

```
You are a contract analysis expert. Your task is to extract the following
information from the provided project charter (which is a PDF):

- "livello_di_rischio" is a string that can only be: "Low", "Medium",
  "High", or "Critical".
- For the numbers, avoid any thousands separators: use 100000 instead of
  100,000. Decimal values should use a dot (e.g., 24.56).
- The value of commercial_estimates must always be greater than
  delivery_estimates. If this is not the case, invert them.
- If commercial_estimates is missing, assume it is 0.0. Then calculate
  delivery_estimates as: commercial_estimates * 0.8.
- Extract the list of technologies mentioned in the document.
- Extract the customer.
```

- Extract the list mesi_lavorati by identifying start_date and end_date (in month format), and reconstruct the ordered list as per the following logic:

```
OrdineMesi = ["gennaio","febbraio","marzo","aprile","maggio","giugno",
              "luglio","agosto","settembre","ottobre","novembre","dicembre"]

indice_inizio = OrdineMesi.index(start_mese)
indice_fine = OrdineMesi.index(end_mese)

if indice_fine < indice_inizio:
    elencoMesi = OrdineMesi[indice_inizio:] + OrdineMesi[:indice_fine + 1]
else:
    elencoMesi = OrdineMesi[indice_inizio:indice_fine + 1]
```

- Translate any roles from Italian to English.

Return your answer as a JSON object with the following structure:

```
{
  "livello_di_rischio" : "string",
  "commercial_estimates": float,
  "delivery_estimates": float,
  "technologies": [string],
  "customer": "string",
  "mesi_lavorati": [string]
}
```

Il prompt è stato progettato per essere robusto rispetto a differenze lessicali e strutturali nei documenti PDF. Il modello impiegato (GPT-4o-mini) è in grado di seguire istruzioni complesse e produrre una risposta coerente con lo schema JSON atteso.

Bibliografia

[1] J. H. Gutiérrez et al. «Integer programming approaches to the multiple team formation problem». In: *Computers & Operations Research* 131 (2021), p. 105267. DOI: 10.1016/j.cor.2021.105267. URL: https://www.sciencedirect.com/science/article/abs/pii/S0305054821001325.

[2] F. Hedberg et al. «Predicting team performance based on past individual achievements using artificial neural networks». In: *PMI Research Conference: Defining the Future of Project Management*. Washington, DC: Project Management Institute, 2010. URL: https://www.pmi.org/learning/library/team-performance-artificial-neural-networks-6494.

[3] Y. Naveh et al. «Workforce optimization: Identification and assignment of professional workers using constraint programming». In: *IBM Journal of Research and Development* 51.3/4 (2007), pp. 359–371. DOI: 10.1147/rd.513.0359. URL: https://www.researchgate.net/publication/224102877_Workforce_optimiztion_Identification_and_assignment_of_professional_workers_using_constraint_programming.

[4] A. Sapienza et al. «Modeling teams performance using deep representational learning on graphs». In: *EPJ Data Science* 13.1 (2024). DOI: 10.1140/epjds/s13688-023-00442-1. URL: https://epjdatascience.springeropen.com/articles/10.1140/epjds/s13688-023-00442-1.

[5] Scikit-learn developers. *sklearn.ensemble.AdaBoostClassifier — scikit-learn 1.2.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html. 2025.

[6] Scikit-learn developers. *sklearn.ensemble.GradientBoostingClassifier — scikit-learn 1.2.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html. 2025.

[7] Scikit-learn developers. *sklearn.ensemble.RandomForestClassifier — scikit-learn 1.2.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. 2025.

[8] Scikit-learn developers. *sklearn.metrics.classification_report — scikit-learn 1.2.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html. 2025.

[9] Scikit-learn developers. *sklearn.metrics.precision_recall_fscore_support — scikit-learn 1.2.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html. 2025.

[10] Scikit-learn developers. *sklearn.neighbors.KNeighborsClassifier — scikit-learn 1.2.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. 2025.

[11] Scikit-learn developers. *sklearn.svm.SVC — scikit-learn 1.2.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. 2025.

[12] Wikipedia contributors. *Assignment problem*. Wikipedia, The Free Encyclopedia. 2025. URL: https://en.wikipedia.org/wiki/Assignment_problem.

[13] XGBoost developers. *xgboost.XGBClassifier — XGBoost 1.7.6 documentation*. 2025. URL: https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier.

[14] Zilliz Inc. *Milvus Documentation*. https://milvus.io/docs. 2024.