Alma Mater Studiorum · Università di Bologna

**SCUOLA DI SCIENZE**
**Laurea magistrale in Informatica**

# UTILIZZO DI MODELLI CONTEXTUAL BANDIT PER L'OTTIMIZZAZIONE DELLE CAMPAGNE PUBBLICITARIE PPC

**Relatore:**
**Chiar.mo Prof.**
**Davide Evangelista**

**Presentata da:**
**Leonardo Marzocchi**

**Sessione**
**Anno Accademico**
**2025 / 2026**

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Introduzione

Negli ultimi anni il commercio elettronico ha conosciuto una crescita esponenziale, trasformando radicalmente le dinamiche di acquisto e vendita a livello globale. In tale contesto, Amazon rappresenta uno degli attori principali, avendo rivoluzionato non solo la distribuzione di prodotti, ma anche le modalità di promozione e pubblicità all'interno della propria piattaforma. Le aziende che scelgono di operare su Amazon devono affrontare un mercato altamente competitivo, caratterizzato da dinamiche algoritmiche complesse e in gran parte non trasparenti agli operatori. Per tale motivo, l'ottimizzazione delle campagne pubblicitarie Pay-Per-Click (PPC) è divenuta una sfida centrale, in quanto consente di bilanciare la visibilità dei prodotti con l'efficienza economica degli investimenti pubblicitari.

Amazon Ads, la piattaforma pubblicitaria interna, mette a disposizione dei venditori una serie di strumenti per la creazione e la gestione di campagne sponsorizzate, tra cui le *Sponsored Products*. Le campagne possono essere gestite tramite strategie di targeting manuale o automatico, facendo leva su keyword e offerte (*bids*), le cui dinamiche di prezzo dipendono da fattori come stagionalità, trend di mercato e eventi promozionali. A supporto di tali attività, Amazon fornisce metriche dettagliate, tra cui impressioni, clic, vendite e indicatori di efficienza come ACOS (Advertising Cost of Sales) e TACOS (Total Advertising Cost of Sales), fondamentali per valutare la redditività delle campagne.

L'analisi quantitativa dei dati rappresenta un passaggio cruciale per migliorare la gestione di tali campagne. In particolare, l'estrazione e il trattamento dei dataset forniti da Seller Central, come il *Sponsored Products Search Term Report* e il *Sponsored Products Advertised Product Report*, consentono di ricostruire le interazioni tra keyword, annunci e prodotti, fornendo la base per lo sviluppo di modelli predittivi. Tuttavia, la natura eterogenea e spesso incompleta dei dati impone un'attenta attività di preprocessing, con la selezione e trasformazione delle variabili più rilevanti. In questo ambito, l'uso di tecniche di rappresentazione semantica delle keyword, come gli embedding generati da modelli linguistici pre-addestrati (ad esempio SBERT), permette di cogliere relazioni latenti tra termini di ricerca e prodotti sponsorizzati, migliorando l'efficacia delle campagne automatiche.

Il cuore metodologico della presente ricerca risiede nell'applicazione di modelli di machine learning per ottimizzare le campagne PPC su Amazon. In una prima fase, sono stati impiegati modelli di regressione classici come LightGBM, XGBoost e Random Forest, al fine di stabilire una baseline solida per le previsioni di metriche chiave (clic, impressioni, spesa e vendite). Successivamente, l'attenzione si è focalizzata su modelli neurali, in particolare le reti Multi-Layer Perceptron (MLP), che hanno dimostrato una maggiore capacità di catturare le relazioni non lineari tra le variabili e di fornire stime più accurate. Parallelamente, sono state condotte sperimentazioni con modelli sequenziali, come LSTM, al fine di gestire le dipendenze temporali presenti nei dati giornalieri.

L'elemento innovativo dell'elaborato è rappresentato dall'integrazione dei modelli pre-

dittivi in un framework di *contextual bandit*, che consente di bilanciare le fasi di esplorazione e sfruttamento nella scelta delle azioni pubblicitarie. Questo approccio, inizializzato con le predizioni delle MLP, permette di aggiornare dinamicamente le stime delle ricompense in base ai dati osservati, selezionando strategie di bidding più efficaci in funzione del contesto. La valutazione sperimentale ha messo in luce l'importanza di una corretta definizione della funzione di reward e di un'attenta calibrazione degli iperparametri, fattori determinanti per garantire stabilità e coerenza nelle prestazioni.

La tesi si articola in cinque capitoli principali. Nel primo capitolo viene introdotto il contesto operativo di Amazon e delle sue campagne pubblicitarie, insieme agli strumenti e alle metriche più utilizzate per monitorare le performance. Il secondo capitolo si concentra sulla descrizione dei dataset e delle metodologie di preprocessing, con particolare attenzione alle tecniche di rappresentazione semantica delle keyword. Nel terzo capitolo vengono presentati i modelli di machine learning sperimentati, dalle regressioni classiche alle reti neurali, fino al contextual bandit, discutendone l'implementazione e il razionale metodologico. Il quarto capitolo analizza criticamente i risultati sperimentali, evidenziando punti di forza e criticità dei modelli proposti. Infine, il quinto capitolo raccoglie le conclusioni, sintetizzando i contributi della ricerca e proponendo sviluppi futuri sia sul piano dei dati che su quello modellistico.

Il contributo di questa ricerca è duplice: da un lato, dimostrare l'applicabilità delle tecniche di machine learning e reinforcement learning in un contesto reale e complesso come quello delle campagne PPC su Amazon; dall'altro, evidenziare le principali sfide e problmatiche ancora aperte, legate alla qualità e quantità dei dati, alla modellizzazione delle strategie aziendali e alla capacità dei modelli di generalizzare in un ambiente competitivo e dinamico. In questo senso, il lavoro si colloca all'intersezione tra ricerca scientifica e applicazioni pratiche, proponendo soluzioni innovative che possono supportare le aziende nell'ottimizzazione dei propri investimenti pubblicitari e contribuire al dibattito accademico sull'uso dell'intelligenza artificiale nel marketing digitale.

# Capitolo 1

# Definizione del contesto

## 1.1 L'Azienda: Compagnia del Benessere e il Marchio Wellbeauty

La Compagnia del Benessere, fondata nel 1997 ad Arezzo da Barbara Goti e Pierluigi Marzocchi, si propone di portare la cosmesi di lusso nel territorio aretino, valorizzando le radici locali dei fondatori. Nel corso degli anni, l'azienda si è distinta nel panorama nazionale, diventando un punto di riferimento per altre imprese italiane nel settore cosmetico. Grazie a una visione orientata alla qualità e all'innovazione, Compagnia del Benessere ha conquistato una clientela sempre più attenta alla sostenibilità ambientale, senza compromettere gli standard di efficacia richiesti dal mercato.

Nel 2024, in risposta alla crescente domanda di prodotti eco-compatibili, i fondatori hanno lanciato il marchio "Wellbeauty: The Eco-Friendly Company". Questo marchio si concentra sulla produzione di cosmetici tra questi, struccanti, creme idratanti e sieri per la ricrescita delle ciglia, con un'attenzione particolare alla sostenibilità e alla qualità. Oltre al centro benessere di proprietà, il principale canale di vendita per i prodotti Wellbeauty è rappresentato dalla piattaforma Amazon, che consente all'azienda di raggiungere un pubblico globale e di promuovere il proprio marchio in un mercato altamente competitivo.

## 1.2 Il Contesto Operativo: Amazon

Fondata nel 1994 come libreria digitale, Amazon Inc. si è rapidamente evoluta in una delle principali piattaforme di e-commerce al mondo, diversificando la propria offerta per includere una vasta gamma di prodotti e servizi. La piattaforma consente alle aziende di caricare, gestire e vendere i propri prodotti, offrendo un'infrastruttura robusta per raggiungere milioni di clienti in tutto il mondo. Le aziende che operano come venditori a proprio marchio su Amazon sono definite "seller" e possono accedere a una piattaforma dedicata, denominata Seller Central (mostrato nello screen in fig. 1.1, raggiungibile all'indirizzo `sellercentral.amazon.com`), per gestire inventario, ordini e strategie di vendita.

Uno degli strumenti più flessibili offerti da Amazon ai venditori è Amazon Ads, una piattaforma pubblicitaria che permette di promuovere i prodotti attraverso campagne mirate. Questo strumento consente ai seller di acquisire maggiore visibilità all'interno

dello store, posizionando i propri prodotti in evidenza in base a specifiche keyword. Una keyword è definita come una parola o una frase digitata da un cliente nella barra di ricerca di Amazon per individuare un prodotto. L'efficacia delle campagne pubblicitarie dipende dalla capacità di selezionare keyword rilevanti e di ottimizzare le strategie di targeting in base al comportamento degli utenti.

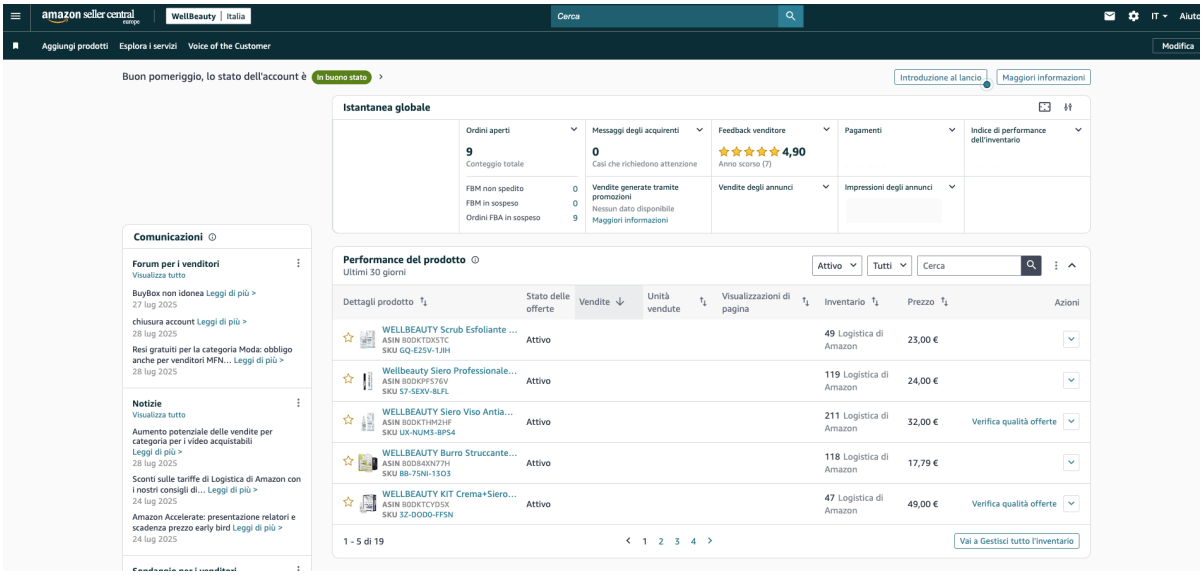


**Figura 1.1:** Dashboard di Amazon seller central per la gestione del profilo venditore.

### 1.2.1 Amazon Ads

Amazon Ads rappresenta una componente fondamentale per le aziende che desiderano massimizzare la visibilità dei propri prodotti sulla piattaforma. Attraverso il portale dedicato, accessibile tramite Seller Central, i venditori possono creare campagne pubblicitarie personalizzate, scegliendo tra diverse tipologie di annunci, come Sponsored Products, Sponsored Brands e Sponsored Display. Ogni campagna può includere più prodotti e adottare diverse strategie di targeting, come il targeting automatico o manuale, per raggiungere specifici segmenti di clientela.

Le campagne Sponsored Products, ad esempio, consentono di promuovere singoli prodotti, posizionandoli in alto nei risultati di ricerca o nelle pagine prodotto correlate. Le campagne Sponsored Brands, invece, sono progettate per aumentare la notorietà del marchio, mostrando annunci che includono il logo aziendale e una selezione di prodotti. Infine, Sponsored Display consente di raggiungere i clienti sia all'interno che all'esterno della piattaforma Amazon, attraverso annunci display basati su interessi o comportamenti di acquisto.

Un aspetto cruciale di Amazon Ads è la possibilità di utilizzare il targeting automatico, che si basa sugli algoritmi di Amazon per associare i prodotti alle ricerche degli utenti, e il targeting manuale, che permette ai venditori di selezionare specifiche keyword per ottimizzare la visibilità. I report generati da Amazon Ads forniscono dati dettagliati sulle prestazioni delle campagne, come il numero di impressioni, clic e conversioni, oltre a informazioni sulle keyword utilizzate dai clienti ("*customer search term*") e sul tipo

di corrispondenza (es. "*loose-match*", "*phrase-match*" o "*exact-match*"). Questi dati sono fondamentali per analizzare il comportamento degli utenti e ottimizzare le strategie pubblicitarie.

Inoltre, Amazon Ads offre strumenti avanzati di analisi, come il Brand Analytics, che consente ai venditori di monitorare le tendenze di ricerca e il comportamento dei clienti. Questi strumenti permettono di identificare le keyword più performanti e di adattare le campagne in tempo reale per massimizzare il ritorno sull'investimento (ROI). Tuttavia, la gestione efficace di queste campagne richiede competenze tecniche per interpretare i dati e implementare strategie basate su analisi approfondite, come quelle proposte in questa tesi attraverso l'uso di modelli di apprendimento automatico. Di seguito verranno definite le possibili campagne pubblicitarie che possono essere condotte su Amazon.

**Sponsored products** permette di sponsorizzare un prodotto facendolo apparire come "*prodotto elencato*" insieme ad altri prodotti simili in base alle keyword ricercate dagli acquirenti. Ogni prodotto che partecipa a questa campagna viene messo in risalto rispetto agli altri mostrandosi in una posizione di rilievo (solitamente tra i primi 4 risultati). Questo tipo di campagna permette, inoltre, di far rientrare l'articolo all'interno una "*scheda prodotto*"[1], tra i "*prodotti consigliati*"[2]. Questo è uno strumento indispensabile per mettere in evidenza (soprattutto nei primi giorni in cui viene pubblicato) ai clienti un nuovo articolo.

**Sponsored brands** viene utilizzata per mettere in risalto il brand mostrandolo in alto o all'interno della pagine che mostra l'elenco dei prodotti ma con una visualizzazione molto più larga e visibile rispetto ad un qualsiasi prodotto. L'obiettivo di questo tipo di campagna è mettere in evidenza il proprio brand rispetto al prodotto, che in questa tipologia è di secondo piano.

**Sponsored display** permette di creare pubblicità dinamiche che possono essere mostrate non solo all'interno del portale ma anche su siti terzi. Questo tipo di campagna dà un vantaggio differente rispetto alle precedenti, mentre le altre appaiono solo quando un utente ricerca il prodotto nel portale, questo tipo di sponsorizzazione è in grado di raggiungere il cliente al di fuori di Amazon stessa.

**Sponsored TV** Offre la possibilità di promuovere i propri prodotti sui servizi di streaming video di proprietà Amazon come: Amazon Prime Video, Twitch, Freevee. Attraverso questa campagna, che al momento della redazione di questa tesi è in fase beta, è possibile raggiungere i clienti attraverso pop-up interattivi in video basandosi anche sul contenuto del video stesso.

[1]La **scheda prodotto** è la pagina Amazon dove l'utente può vedere le caratteristiche e le immagini dell'articolo che intende acquistare. In questa pagina il cliente, oltre a vedere descrizione e recensioni, può anche contattare il marchio venditore per porre pubblicamente domande inerenti all'articolo mostrato.

[2]I **prodotti consigliati** è un elenco di articoli alternativi a quello che si sta visualizzando nella *scheda prodotto* volto a indirizzare il cliente ad una scelta più ampia di merce.

**La strategia**

una strategia è un metodo che definisce come una determinata campagna proporrà il prodotto oggetto della campagna agli utenti che effettuano la ricerca. Ogni strategia è standardizzata e necessita di:

- Una parola o una sequenza di parole (Keyword)
- Una offerta (bid) per ogni keyword

Per ogni keyword il bid può essere impostato automaticamente, seguendo quindi il criterio di costo definito da Amazon internamente per quella keyword, oppure inserito manualmente. In base alla quantità di ricerche della quale una keyword gode, Amazon aumenta il prezzo consigliato per quella keyword. Il prezzo consigliato è un valore suggerito da Amazon che viene calcolato internamente; nel caso di scelta automatica del bid, questo andrà a seguire il suddetto valore. In base ai successivi flussi di ricerca, o in base a determinati eventi promozionali (quali "Black Friday","Prime Day","Offerte di Natale", ecc...) il prezzo del bid aumenterà o diminuirà. Ogni strategia può intercettare una ricerca in base alla sua natura; di seguito vediamo le varie strategie e gli esempi che sono in grado di catturare. La keyword all'interno delle campagne sponsorizzate assume il nome di "Target", quando un utente ricerca una parola Target, quella sua ricerca assume il nome di "Customer Search Term"

**Tabella 1.1:** Esempi di strategie di match

| tipo di strategia | Default keyword | esempi di customer search term |
|---|---|---|
| generica | "Zaino" | "Borsa da ufficio" |
| a frase | "Collare cane" | "Collare per cani di piccola taglia" |
| esatta | "Auricolari Bluetooth" | "Auricolari Bluetooth" |

- Corrispondenza Generica: determinata la parola Target il match può essere composto con una sequenza semanticamente simile alla keyword, dove la sequenza di parole può essere anche completamente diverse ma hanno lo stesso significato semantico.
- Corrispondenza a Frase: è un tipo di ricerca più rispetto a quella generica, il match può avvenire con frasi che includo la parola target, ma il customer search term deve comprendere almeno tutte le parole nel target. Avviene un match anche nel caso in cui venga ricercato quel termine plurale o singolare oppure se sono presenti preposizioni semplici.
- Corrispondenza esatta: viene effettuato il match solo se la ricerca contiene le esatte parole presenti nel target. Alcune leggere varianti della parola sono concesse.

Il prezzo del bid, come precedentemente detto, può essere inserito manualmente. Quando viene deciso un prezzo per il bid, è bene tenere in considerazione diversi fattori:

- Andamento di mercato: Una keyword è soggetta ai vari andamenti di mercato determinati da:

  - Andamenti stagionali: ci sono articoli che sono sono soggetti alla stagione nella quale vengono utilizzati.Un esempio rappresentativo è costituito dagli occhiali da sole, la cui domanda è fortemente influenzata dalla stagionalità. In inverno è raro che ci siano persone che cerchino occhiali da sole, mentre in estate il volume di ricerche si intensifica e con esso anche il prezzo per quelle keyword aumenta.
  - Tendenze: sono prodotti che improvvisamente godono di una certa fama, la durata del trend è variabile e non se ne conosce quasi mai la durata. I prodotti in tendenza hanno un costo del bid maggiore rispetto a prima che fossero tendenze

- Giornate o periodi di offerta promossi da Amazon: generalmente periodi come il Black Friday sono momenti di grandi afflussi di clienti. Essendo periodi di tempo che attraggono il cliente grazie agli sconti che vengono proposti, i venditori sono propensi a creare sconti maggiori ed a aumentare il prezzo del bid per permettere ai clienti di farli trovare più facilmente.

- strategie interne all'azienda: In base alle strategie che un'azienda decide di perseguire viene cambiato il prezzo. Prossimamente andremo a vederle in maniera più dettagliata.

- Prezzo suggerito: il prezzo suggerito da Amazon è un valore che permette al prodotto pubblicizzato di apparire in posizioni rilevanti all'interno della pagina. Al di sopra di quel prezzo è sempre più garantita una posizione di rilievo, mentre se si scendesse sotto il prezzo proposto il prodotto potrebbe non apparire proprio tra le sponsorizzazioni.

**Le strategie aziendali**

Ogni azienda che lavora all'interno della piattaforma Amazon come Seller può adottare delle strategie per il raggiungimento di un obiettivo. Prima però di definire le varie strategie è importante dare la definizione di "*posizionamento organico*".
Il **Posizionamento organico** è il posizionamento di cui gode un prodotto all'interno della categoria alla quale afferisce. Questa posizione ha un impatto rilevante all'interno della pagina di ricerca. Il prodotto viene posizionato da Amazon in base alle vendite generate, se sono più alte (degli altri prodotti nella categoria) il prodotto apparirà per primo; al calare delle vendite scenderà nel posizionamento.
Un esempio pratico è il seguente: se un cliente effettua una ricerca del prodotto "Tastiera da gaming" e un articolo, che non appare come sponsorizzato al momento della ricerca, si posiziona come Terzo elemento all'interno della pagina, questo avrà come posizionamento organico pari a 3. Se il cliente cliccasse sul suddetto articolo che appare senza il simbolo "sponsorizzato" al venditore non verrebbe addebitato nessun costo, al contrario se cliccasse sullo stesso prodotto ma con apposto il simbolo "sponsorizzato" il venditore dovrà sostenere il costo del click per quella keyword.

Definito il significato di Posizionamento organico, si può procedere a capire come si articolino le strategie aziendali. Generalemente le strategie sono le seguenti:

- **Branding**: vengono utilizzate le pubblicità per apparire in prima pagina, il prezzo dei bid è molto alto molto spesso le vendite vengono effettuate in perdita pur

di mostrare il proprio prodotto come primo. Richiede fondi molto elevati per la maggior parte dei prodotti. Questa è una strategia che si attua quando si cerca di aumentare la visibilità dei propri prodotti

- **Ranking**: Si decide di aumentare i bid delle keyword scegliendo eventualmente anche le keyword con molto traffico e molto costose con l'obiettivo di promuovere il prodotto e venderlo, non è importate generare del profitto ma è importante effettuare tante vendite. Se questa strategia funziona, il prodotto risulterà più in alto nelle classifiche permettendo al venditore di essere vistato senza dover pagare per le sponsorizzate nei successivi click. Questa è una strategia che si percorre solitamente durante il lancio di un nuovo prodotto.

- **Shield**: è una strategia che permette al venditore di proteggersi da altri competitor. All'interno della pagina del suo prodotto fa in modo di far apparire prodotti sponsorizzati solo del suo brand.

- **Research**: Questa strategia viene affrontata per effettuare indagini di mercato e capire se un segmento è appropriato a quell'azienda oppure no. Normalmente in questa strategia (che può anche essere vista come una fase pubblicitaria) vengono considerati costi pubblicitari e vengono stimati i margini di profitto del prodotto che si intende vendere.

- **Profit**: questa strategia cerca di minimizzare i costi pubblicitari cercando di generare vendite, l'obiettivo è quello di spendere meno possibile per vendere un prodotto

Nel contesto delle campagne pubblicitarie su Amazon, due indicatori fondamentali per la valutazione dell'efficacia della spesa pubblicitaria sono l'**ACOS** (Advertising Cost of Sales) e il **TACOS** (Total Advertising Cost of Sales). Di seguito si riportano le definizioni formali di entrambe le metriche:

$$\text{ACOS} = \frac{\text{SP}_{\text{adv}}}{\text{SA}_{\text{adv}}} \times 100 \tag{1.1}$$

Dove:

- $\text{SP}_{\text{adv}}$ = Totale della spesa pubblicitaria effettuata nelle campagne.

- $\text{SA}_{\text{adv}}$ = Totale delle vendite attribuite direttamente alle campagne pubblicitarie.

L'ACOS misura l'incidenza della spesa pubblicitaria rispetto ai ricavi generati direttamente dalle sponsorizzazioni. Non tiene conto delle vendite ottenute in modo organico.

$$\text{TACOS} = \frac{\text{SP}_{\text{adv}}}{\text{SA}_{\text{acc}}} \times 100 \tag{1.2}$$

Dove:

- $\text{SP}_{\text{adv}}$ = Totale della spesa pubblicitaria effettuata nelle campagne (come sopra).

- $\text{SA}_{\text{acc}}$ = Totale delle vendite generate dall'intero account, comprensive sia di vendite pubblicitarie che di vendite organiche.

Il TACOS fornisce una visione più ampia dell'impatto della pubblicità sull'intero volume di vendite, permettendo di valutare anche l'effetto indiretto della pubblicità sul posizionamento organico e sul brand awareness.

Questo indicatore è indicativo dell'andamento globale dell'account, misurando le spese percentuali di ogni prodotto si ottiene un quadro completo delle spese pubblicitarie e per questo si può dedurre un andamento delle strategie che si stanno perseguendo.

In base alle categorie del prodotto che si intende vendere il costo pubblicitario varia molto, in determinate categorie come quella dei cosmetici può senza difficoltà arrivare anche al 50% del costo del prodotto.

**Alcune dinamiche riguardo al prezzo**

Ogni venditore non è in grado di poter vedere il prezzo del bid che pagano gli altri venditori. Guardando il bid proposto da Amazon si può avere un'indicazione del prezzo di mercato di quella keyword. Ogni venditore come non consce il prezzo degli altri prodotti non può nemmeno conoscere il costo pubblicitario che gli altri venditori sostengono. Un venditore può sapere quanti prodotti di altri venditori sono stati venduti grazie all'utilizzo di applicazioni apposite (come Helium10, AmzScout, ecc...)

## 1.3 Introduzione al problema

Nel contesto competitivo del mercato digitale, la pubblicità online rappresenta uno degli strumenti fondamentali attraverso cui le imprese cercano di intercettare la domanda e orientare le decisioni d'acquisto dei consumatori.
In tale scenario, emerge una problematica strategica rilevante: come ottimizzare l'investimento pubblicitario minimizzando la spesa associata al bidding, senza compromettere l'efficacia della campagna in termini di vendite e profitti? La sfida è quella di trovare un equilibrio ottimale tra due obiettivi potenzialmente in conflitto: ridurre i costi pubblicitari e, contemporaneamente, massimizzare i ricavi derivanti dalle conversioni generate dagli annunci.

Questo elaborato si propone di affrontare il problema mediante un approccio quantitativo, sviluppando e analizzando modelli che permettano di regolare dinamicamente il prezzo del bid in funzione delle performance ottenute, dei costi sostenuti e degli obiettivi economici prefissati. In particolare, si mira a identificare strategie di ottimizzazione capaci di adattarsi ai cambiamenti del mercato e del comportamento degli utenti, contribuendo così a migliorare l'efficienza complessiva delle campagne pubblicitarie online.

# Capitolo 2

# Metodologia

L'obiettivo principale di questo elaborato è analizzare le statistiche di mercato per i prodotti presi in esame, verranno utilizzati i risultati di questa analisi al fine di trovare il prezzo minimo per la pubblicità che massimizzi il profitto. Data una keyword analizzeremo come estrarne il valore semantico e come questo influisce nelle metriche di vendita.

## 2.1 Analisi dei dati

In questa sezione verranno analizzati i dati che sono stati acquisiti e come questi dati verranno processati al fine di ottenere il miglior modello possibile al fine di massimizzare il profitto minimizzando l'ACOS.

### 2.1.1 Acquisizione dei dati

Amazon mette a disposizione dei venditori la piattaforma Seller Central ("*sellercentral.amazon.com*), questa è una dashboard utilizzata da un venditore per monitorare l'andamento delle vendite dei propri prodotti, gestire l'inventario, ottenere le metriche relative alla salute del proprio account (ex recensioni, opinioni del cliente, resi, ecc...) (raffigurata in figura 1.1). Le pubblicità in Amazon vengono gestite con un portale a parte introdotto nel capitolo 1.2.1. Questo portale oltre ad essere d'aiuto per il venditore al fine di gestire le sue campagne grazie a grafici semplici e intuitivi consente all'utente di esportare report dettagliati delle performance pubblicitarie. Attraverso lo strumento di reportistica (fig. 2.2) è possibile gestire diversi tipi di report con una granularità di campionatura che varia in base al tipo di dato richiesto. Lo strumento "misurazione e reportistica" è in grado di creare report per tutte le tipologie di pubblicità che è possibile fare (Sponsored products, sponsored brands, sponsored display,sponsored TV). In questo studio analizzeremo solo le pubblicità sotto la tipologia "sponsored products" in quanto non si hanno sufficienti dati delle altre campagne per condurre un'analisi significativa dal punto di vista statistico.

Per eseguire il report delle campagne si può scegliere di farlo attraverso una campionatura di tipo "riepilogo" i cui dati vengono riassunti in un breve riepilogo mensile (o dei giorni scelti attraverso il selezionatore della data) oppure una campionatura giornaliera più dettagliata; in questo caso otterremo un file in formato *.xlsx* (Microsoft Excel) contenente tutte le keyword di ogni campagna. In questo caso è stato effettuato il report

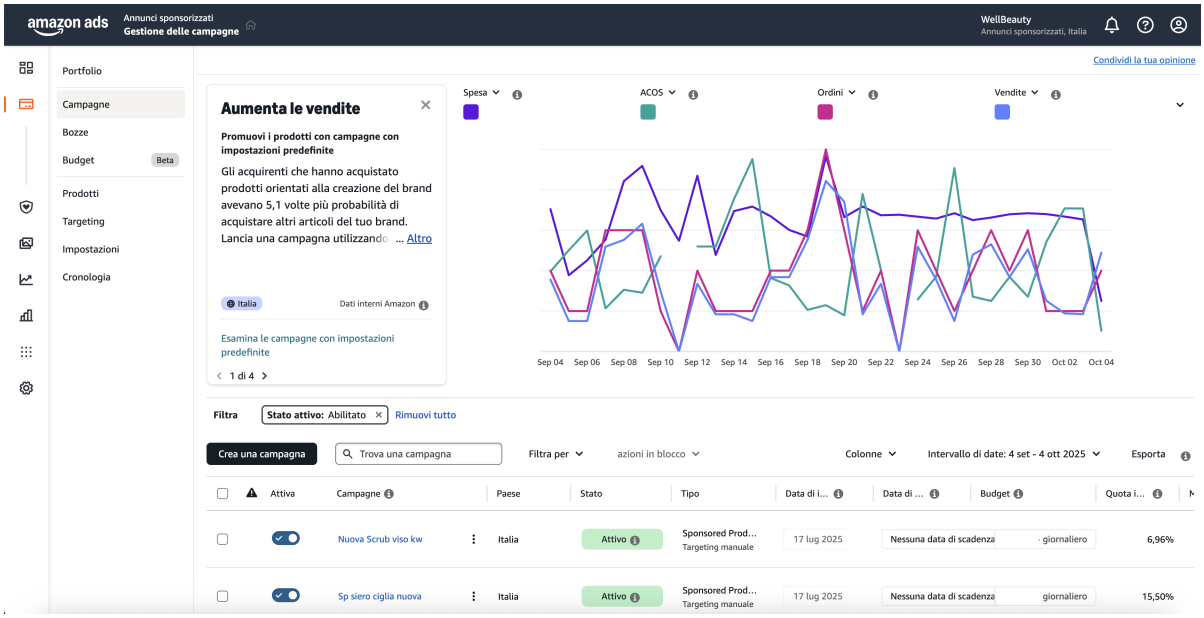


**Figura 2.1:** Dashboard di Amazon ads per la gestione delle campagne pubblicitarie.

giornaliero delle campagne pubblicitarie di un periodo di 30 giorni. La reportistica con granularità giornaliera permette di ottenere delle misurazioni sufficientemente dettagliate per comprendere variazioni e andamenti di periodo (settimanali e giornalieri).

Lo strumento "misurazioni e reportistica" offre misurazioni discrete dell'andamento delle campagne, nessuno dei dati riportati è frutto di algoritmi probabilistici, tutti i dati sono deterministici perciò viene utilizzato anche come strumento con valenza legale, ciò significa che è richiesto che i dati siano accurati e non vi siano errori in quanto vengono utilizzati a sua volta questi dati per fatturare le spese pubblicitarie agli utenti venditori [2]. In questi studi verrà preso come assunto iniziale la correttezza di questi dati e di conseguenza non verranno utilizzate strategie per la rimozione di possibili errori nella campionatura.

### 2.1.2 Descrizione dei dataset

I dataset per questo esperimento sono due:

- *Sponsored Products Search term report*
- *Sponsored Product advertised product report*

#### Sponsored Products Search term report

Questo dataset descritto nella tabella 2.1 è un report che elenca attraverso quali keyword gli utenti hanno raggiunto un determinato prodotto; questo report è fondamentale sia per un aspetto aziendale, in quanto incapsula il quadro generale di come il cliente finale trova un determinato prodotto sia per fini di ricerca in quanto non potendo prevedere tutti i termini con la quale un cliente trova i prodotti possiamo analizzare quelli che hanno già ricercato.

**Tabella 2.1:** Descrizione delle colonne presenti nel dataset di Amazon Sponsored Products Customer Search Term. La tabella riassume il significato di ciascun campo.

| **Colonna** | **Descrizione** |
|---|---|
| Date | Data di registrazione dei dati relativi alla campagna pubblicitaria. |
| Portfolio name | Nome del portfolio che aggrega più campagne pubblicitarie. |
| Currency | Valuta utilizzata per esprimere gli importi monetari (es. USD, EUR). |
| Campaign Name | Nome identificativo della specifica campagna. |
| Ad Group Name | Nome del gruppo di annunci all'interno della campagna. |
| Targeting | Parola utilizzata per catturare Customer search term (può essere automatica indicata con il simbolo '*' o manuale). |
| Match Type | Tipo di corrispondenza tra keyword e termine di ricerca (broad, phrase, exact). |
| Customer Search Term | Termine di ricerca digitato dal cliente su Amazon. |
| Impressions | Numero totale di visualizzazioni dell'annuncio. |
| Clicks | Numero di clic ricevuti sull'annuncio. |
| Click-Thru Rate (CTR) | Percentuale di clic sul totale delle impressioni. |
| Cost Per Click (CPC) | Costo medio sostenuto per ciascun clic. |
| Spend | Spesa pubblicitaria totale sostenuta. |
| 7 Day Total Sales | Vendite totali generate entro 7 giorni dal clic sull'annuncio. |
| Total Advertising Cost of Sales (ACOS) | Percentuale che indica il rapporto tra spesa e vendite (più basso è, meglio è). |
| Total Return on Advertising Spend (ROAS) | Ritorno sull'investimento pubblicitario (valore delle vendite diviso spesa). |
| 7 Day Total Orders (#) | Numero complessivo di ordini effettuati entro 7 giorni dal clic. |
| 7 Day Total Units (#) | Numero totale di unità vendute entro 7 giorni. |
| 7 Day Conversion Rate | Tasso di conversione (ordini/clic) entro 7 giorni. |
| 7 Day Advertised SKU Units (#) | Unità vendute del prodotto sponsorizzato. |
| 7 Day Other SKU Units (#) | Unità vendute di altri prodotti non sponsorizzati direttamente. |
| 7 Day Advertised SKU Sales | Vendite (in valore) generate dal prodotto sponsorizzato. |
| 7 Day Other SKU Sales | Vendite (in valore) generate da altri prodotti dopo il clic. |

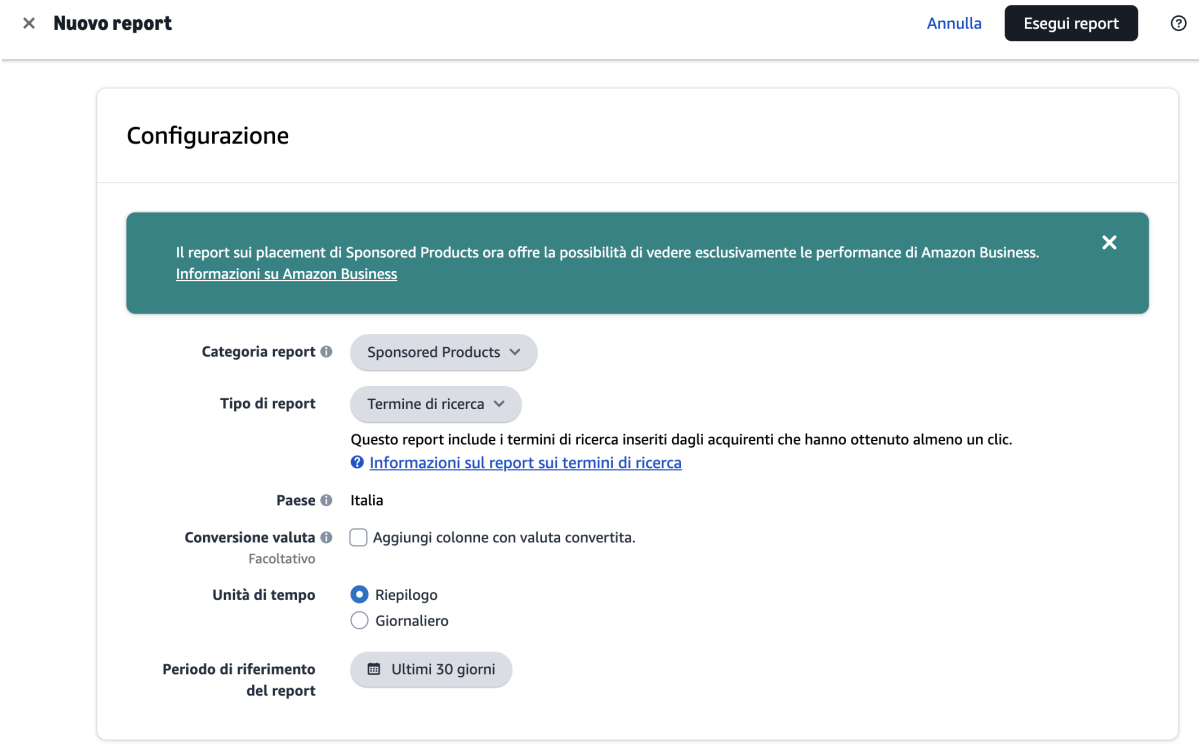


**Figura 2.2:** Dashboard di Amazon per la creazione di report. Oltre alle impostazioni di report mostrate in figura è possibile applicare altre impostazioni più specifiche

### Sponsored Product advertised product report

Il dataset descritto nella tabella 2.1 è un consuntivo che definisce a quale prodotto ogni campagna è collegato; Dal momento che il primo report riporta solo i dati relativi alle statistiche delle varie keyword giornalmente, ad ogni keyword non viene collegato nessun identificativo.

**Tabella 2.2:** Descrizione delle colonne del dataset Amazon Sponsored Products in lingua italiana, usato per analizzare performance pubblicitarie localizzate per paese e SKU.

| **Colonna** | **Descrizione** |
|---|---|
| Data | Data in cui è stato registrato il dato della campagna. |
| Nome portafoglio | Nome del portafoglio che raccoglie più campagne. |
| Valuta | Valuta in cui sono espresse le spese e vendite. |
| Nome campagna | Nome della specifica campagna pubblicitaria. |
| Nome gruppo di annunci | Nome del gruppo di annunci all'interno della campagna. |
| Paese | Codice paese del marketplace Amazon (es. IT, DE, FR). |
| SKU pubblicato | Codice SKU del prodotto sponsorizzato. |
| ASIN pubblicato | Codice ASIN del prodotto sponsorizzato. |
| Impressioni | Numero di volte in cui l'annuncio è stato visualizzato. |
| Clic | Numero di clic ricevuti sull'annuncio. |
| Percentuale di accesso (CTR) | Percentuale di clic rispetto alle impressioni. |
| Costo per clic (CPC) | Costo medio pagato per ogni clic. |
| Spesa | Spesa totale sostenuta per l'annuncio. |
| Vendite totali (€) 7 giorni | Vendite totali generate entro 7 giorni dal clic. |
| Costo pubblicitario delle vendite (ACOS) totale | Rapporto tra spesa pubblicitaria e vendite totali. |
| Ritorno sulla spesa pubblicitaria (ROAS) totale | Rapporto tra vendite e spesa pubblicitaria. |
| Totale ordini (#) 7 giorni | Numero totale di ordini ricevuti entro 7 giorni. |
| Totale unità (#) 7 giorni | Numero totale di unità vendute in 7 giorni. |
| Tasso di conversione 7 giorni | Percentuale di conversione tra clic e ordini entro 7 giorni. |
| Unità SKU pubblicizzati 7 giorni (#) | Unità vendute del prodotto sponsorizzato. |
| Altre unità SKU (#) 7 giorni | Altre unità vendute non direttamente sponsorizzate. |
| Vendite SKU pubblicizzati 7 giorni (€) | Valore delle vendite generate dal prodotto sponsorizzato. |
| Vendite altro SKU (€) 7 giorni | Valore delle vendite di altri prodotti dopo il clic. |

## 2.2 Preprocessing

In questa sezione si approfondisce come i dati precedentemente acquisiti vengono processati al fine di ottenere il dataset più adatto per raggiungere l'obiettivo di ottimizzazione del prezzo per un determinato bid.

### 2.2.1 Unione dei dataset

Nel dataset *Customer Search Term* (Tab. 2.1) – da ora in poi indicato con la sigla **CST** – sono riportati, su base giornaliera, i termini di ricerca (*search terms*) utilizzati dagli utenti per raggiungere i prodotti sponsorizzati, insieme alla strategia di targeting corrispondente (si veda anche Tab. 1.1).

Tuttavia, una criticità strutturale di questo dataset consiste nell'assenza di un identificativo univoco del prodotto (ad esempio SKU o ASIN) associato a ciascun termine di ricerca. Tale mancanza impedisce di contestualizzare i dati rispetto al singolo prodotto sponsorizzato. In ambito pubblicitario, questo aspetto è particolarmente rilevante, poiché il valore di un *search term* può variare sensibilmente a seconda delle caratteristiche del prodotto a cui è collegato. La definizione di una strategia di offerta (*bidding*) ottimale per ciascuna parola chiave dipende, infatti, non solo dalle metriche aggregate di performance, ma anche dalla coerenza semantica tra il termine di ricerca e le specificità del prodotto promosso.

A titolo esemplificativo, si consideri un venditore che sponsorizza due prodotti per la pulizia dei pavimenti:

- Prodotto **A**: detersivo con profumazione leggera, ma con elevato potere igienizzante.
- Prodotto **B**: detersivo con composizione delicata, ma con una fragranza molto intensa.

Entrambi i prodotti potrebbero risultare pertinenti per la keyword generica *"detersivo pavimenti profumato"*. Tuttavia, dal punto di vista strategico, il venditore potrebbe essere disposto ad attribuire un'offerta (*bid*) più elevata alla keyword quando questa è associata al prodotto **B**, in quanto la caratteristica predominante richiesta dall'utente (la profumazione) è meglio rappresentata da tale articolo.

In assenza di un collegamento diretto tra il termine di ricerca e il prodotto di destinazione, tale distinzione non è possibile, con il risultato che le decisioni di ottimizzazione dei bid devono basarsi su medie aggregate o assunzioni, riducendo l'efficacia complessiva della strategia PPC.

Pertanto, si propone l'integrazione dell'identificativo del prodotto nel dataset **CST** come elemento essenziale per aumentare il grado di granularità e precisione nelle strategie di ottimizzazione delle offerte, con l'obiettivo di massimizzare il ritorno sull'investimento pubblicitario (ROAS) e ridurre il costo pubblicitario delle vendite (ACOS).

Il problema che si pone in questo stadio è quando per una campagna pubblicitaria vengono rappresentati più di due prodotti e di conseguenza le keyword hanno lo stesso prezzo (ma non hanno necessariamente tutte le altre metriche uguali come per esempio

Click, impression, ecc...). Dal momento che è impossibile discriminare le caratteristiche come impression, click e sales per i prodotti contenuti all'interno della stessa campagna si opta per un *merge left*. Per ogni tupla di una campagna con più prodotti collegati all'interno del dataset *Sponsored product advertised product* vengono clonate le relative tuple presenti nel dataset *Sponsored product search term* fino al raggiungimento del numero del primo dataset.

In quest'ottica si procede con l'unione dei due dataset in uno unico. L'obiettivo di questa unione è collegare ad ogni keyword il prodotto al quale si fa riferimento. Il primo passo per unire i due dataset è quello di trovare le chiavi comuni, queste sono:

- *Date*
- *Portfolio Name*
- *Currency*
- *Campaign Name*
- *Ad group name*

### 2.2.2 Pulizia delle feature

Passo importante per ottenere un dataset di buona qualità è quello della pulizia, in principio si rimuovono le feature superflue e successivamente si passa ad un'analisi più accurata per capire quali feature mantenere.

Successivamente all'unione del dataset riportata nella sezione precedente si mantiene solo la colonna "*ASIN*" che indica l'identificativo univoco all'interno del dataset; questo viene fatto per non avere colonne ridondanti (praticamente tutte le colonne indicano la stessa tipologia di dato ma raggruppata con criteri differenti). Il dataset risultante dall'unione di *Sponsored product search term* e *Sponsored product advertised product* è quello sulla quale si opererà da questo momento in poi, pertanto si chiamerà, da adesso *dataset principale*. Dobbiamo identificare nel dataset principale le feature non rilevanti al fine del problema, per questo si decide di eliminarle in base a due strategie:

- **irrilevanza** ai fini dell'obiettivo
- **metriche calcolate** o composte da altre colonne

Tra le metriche irrilevanti possiamo trovare:

- *Currency*: Wellbeauty al momento dell'utilizzo del dataset lavora solo in italia e la decisione del bid avviene sempre e solo in Euro
- *Portfolio Name*: definisce il nome del portafoglio che contiene altre campagne pubblicitarie
- *7 Day Total Orders (#),7 Day Total Units (#),7 Day Advertised SKU Units (#),7 Day Other SKU Units (#),7 Day Advertised SKU Sales,7 Day Other SKU Sales*: sono metriche numeriche che indicano la conversione in ordini, quello che ci interessa sono le vendite espresse in valore economico in quanto il valore di un prodotto

può variare da un giorno all'altro e per periodi non definiti (gli sconti possono durare da qualche ora a qualche giorno). Inserirli come variabili di target del modello aggiungerebbero complessità crescente senza aggiungere nessun beneficio al risultato del modello.

Di seguito sono riportate le principali metriche calcolate nei report pubblicitari Amazon. Tali metriche non sono fornite come dati grezzi, ma vengono derivate a partire da variabili fondamentali (es. click, impression, costi, vendite) attraverso formule deterministiche. Sono comunemente utilizzate per valutare l'efficacia delle campagne.

- **Click-Thru Rate (CTR)**: rappresenta la percentuale di utenti che, dopo aver visualizzato l'annuncio (impression), hanno effettivamente cliccato su di esso.

$$\text{CTR} = \frac{\text{Clicks}}{\text{Impressions}} \times 100$$

  Una CTR elevata indica un annuncio rilevante o ben posizionato.

- **Cost Per Click (CPC)**: indica il costo medio sostenuto per ogni click ricevuto sull'annuncio.

$$\text{CPC} = \frac{\text{Spend}}{\text{Clicks}}$$

  È una metrica utile per analizzare l'efficienza economica del traffico generato.

- **Total Advertising Cost of Sales (ACOS)**: misura il rapporto tra la spesa pubblicitaria e le vendite attribuite alla pubblicità.

$$\text{ACOS} = \frac{\text{Spend}}{\text{7 Day total sales}} \times 100$$

  Un valore più basso indica maggiore efficienza della campagna in termini di ritorno.

- **Total Return on Advertising Spend (ROAS)**: rappresenta l'inverso dell'ACOS ed esprime il ritorno economico per ogni unità monetaria investita in pubblicità.

$$\text{ROAS} = \frac{\text{7 Day Total Sales}}{\text{Spend}}$$

  Valori elevati di ROAS indicano un buon ritorno sull'investimento pubblicitario.

- **7 Day Conversion Rate**: misura la percentuale di click che hanno portato ad almeno una vendita entro 7 giorni.

$$\text{Conversion Rate} = \frac{\text{7 Day Total Orders}}{\text{Clicks}} \times 100$$

  Una conversione può essere intesa come l'acquisto di un prodotto dopo il click sull'annuncio.

Al termine di questa pulizia delle colonne il nuovo dataset finale sarà composto dalle seguenti colonne:

- **Date**
  Indica la data di riferimento del dato riportato. Ogni riga del dataset corrisponde a una specifica data.

- **Portfolio name**
  Nome del portafoglio pubblicitario, utilizzato per organizzare e raggruppare campagne all'interno dell'account del venditore.

- **Campaign Name**
  Nome assegnato alla campagna pubblicitaria. Serve per identificare e distinguere le varie campagne attive.

- **Ad Group Name**
  Nome del gruppo di annunci associato alla campagna. Ogni campagna può contenere più gruppi di annunci, ciascuno con specifiche keyword e offerte.

- **Targeting**
  Indica la parola chiave o il prodotto target dell'annuncio. Nel caso di targeting manuale, rappresenta la keyword impostata dal venditore.

- **Match Type**
  Specifica la modalità di corrispondenza della keyword:
  - *Broad (generica)* – corrispondenza ampia e flessibile
  - *Phrase (a frase)* – corrispondenza parziale e ordinata
  - *Exact (esatta)* – corrispondenza solo se la ricerca è identica alla keyword

- **Customer Search Term**
  La query di ricerca digitata effettivamente dall'utente. Può differire dalla keyword target in base al tipo di corrispondenza.

- **Impressions**
  Numero di volte in cui l'annuncio è stato visualizzato dagli utenti nei risultati di ricerca.

- **Clicks**
  Numero totale di click ricevuti sull'annuncio. Indica l'interesse suscitato nei confronti dell'inserzione.

- **Spend**
  Totale della spesa pubblicitaria (in euro o nella valuta dell'account) sostenuta per la keyword o il termine di ricerca in quella data.

- **7 Day Total Sales**
  Valore totale delle vendite attribuite alla pubblicità entro 7 giorni dal click sull'annuncio. Include solo le conversioni avvenute entro la finestra temporale indicata.

### 2.2.3 Estrazione della strategia di una campagna

Ogni campagna al momento dell'ideazione viene creata seguendo le quattro campagne riportate nel paragrafo 1.2.1. A seconda della campagna pubblicitaria scelta, il prezzo pubblicitario (e di riflesso anche il suo budget) possono variare notevolmente; è fondamentale incapsulare all'interno del dataset questa feature, che però non è presente all'interno del dataset come colonna. Durante la fase di creazione della campagna si pensa alla strategia da perseguire in base a questa si assegnano i nomi alle campagne:

**Tabella 2.3:** Descrizione di esempio nome campagna: SP—M—KW—Frase—B0D84XN77H—Lancio

| Sponsored Products | M | Keyword | Frase | B0D84XN77H | Lancio |
|---|---|---|---|---|---|
| Tipologia campagna | Manuale o Automatica | Tipo di target impostato | Tipo di corrispondenza con la keyword | ASIN del prodotto sponsorizzato | Obiettivo della strategia aziendale |

In questa campagna viene riportato il termine "**Lacio**" il che identifica che questa campagna è stata creata appositamente per lanciare un prodotto. Il lancio di un prodotto subisce inevitabilmente dei costi più alti rispetto ad un prodotto già avviato in quanto deve essere mostrato ad un gruppo di clienti più ampio possibile. Come questa campagna, anche altre campagne riportano la parola "*lancio*" all'interno del loro nome, ci sono ulteriori campagne invece che riportano il termine "Performance" o "Top Keword"; queste campagne indicano strategie dove il prodotto è stato già recepito dal mercato e quindi la campagna gestisce keyword ad alto traffico (ovvero con molte impression). Queste caratteristiche inglobano di per se delle informazioni importanti che è bene esplicitare nel dataset.

**Estrazione di caratteristiche di performance:** Durante la fase di preprocessing, è stato analizzato il nome di ciascuna campagna pubblicitaria al fine di estrarre informazioni semantiche utili a identificare l'intento strategico della campagna. In particolare, si è effettuata una ricerca all'interno delle stringhe di testo per identificare la presenza di determinate parole chiave ritenute indicatori di performance o strategia:

- Se all'interno del nome della campagna è presente la stringa `"Top keyword"` oppure `"Alto traffico"`, si attribuisce valore `1` alla colonna binaria **performance**, altrimenti `0`. Questo processo equivale a un'operazione di *one-hot encoding* su indicatori lessicali di rilevanza o traffico elevato.
- Se nel nome della campagna compare il termine `"Lancio"`, viene attribuito valore `1` alla colonna **lancio**, indicando che la campagna è probabilmente legata alla fase iniziale di promozione di un prodotto. In assenza di tale parola, viene assegnato valore `0`.

Questa trasformazione consente di convertire informazioni testuali qualitative in feature numeriche binarie, rendendole utilizzabili nei modelli di machine learning impiegati nelle fasi successive.

### 2.2.4 Embedding delle keyword

Nel contesto dell'elaborazione del linguaggio naturale, il termine *embedding* si riferisce al processo di trasformazione di parole o frasi in rappresentazioni numeriche dense (vettori) all'interno di uno spazio continuo a bassa dimensionalità. Questa rappresentazione consente di catturare relazioni semantiche e sintattiche tra le parole, facilitando l'utilizzo di modelli matematici e algoritmi di machine learning su dati testuali.

Nel caso specifico delle keyword presenti nel dataset, l'embedding ha l'obiettivo di rappresentare ciascuna keyword come un vettore numerico che conserva la struttura semantica latente del linguaggio. Due parole con significato simile, ad esempio, avranno rappresentazioni vettoriali vicine nello spazio degli embedding.

Sono stati sviluppati diversi metodi per ottenere embedding, tra cui:

- **Word2Vec** [35]: un modello predittivo che apprende rappresentazioni vettoriali ottimizzando la previsione del contesto di una parola (skip-gram) o della parola centrale dato il contesto (CBOW).

- **GloVe (Global Vectors)** [41]: un modello basato sulle co-occorrenze globali delle parole in un corpus, che ottimizza una funzione di costo che tiene conto delle probabilità con cui le parole appaiono insieme.

- **BERT (Bidirectional Encoder Representations from Transformers)** [14]: un modello di embedding contestuale che utilizza un'architettura Transformer bidirezionale per generare rappresentazioni dinamiche delle parole, che variano in base al contesto.

- **SBERT (Sentence-BERT)** [42]: un'estensione di BERT ottimizzata per la rappresentazione di intere frasi o sequenze di parole, utile per compiti come il clustering, la ricerca semantica e la riduzione dimensionale.

**Scelta dell'algoritmo di embedding** La selezione dell'algoritmo di embedding più adatto rappresenta una fase cruciale per lo sviluppo del sistema di raccomandazione nel contesto delle campagne pubblicitarie PPC. In particolare, si è posta l'attenzione sul problema della *validazione* dell'embedding, ossia sulla valutazione della sua efficacia in un contesto *non supervisionato*. In assenza di etichette di riferimento, diventa complesso determinare quale rappresentazione semantica delle keyword sia più adatta al task.

Una prima strategia consiste nell'effettuare una comparazione empirica tra differenti modelli di embedding, come ad esempio TF-IDF, FastText e SBERT. Tale confronto richiede però un effort computazionale e metodologico considerevole, ed è spesso soggetto a variabilità che ne limita la riproducibilità. Inoltre, molte metriche utilizzabili per il confronto (es. similarità coseno su keyword) non riescono a cogliere correttamente il comportamento semantico in presenza di parole polisemiche o espressioni ambigue.

Si è quindi deciso di selezionare l'algoritmo di embedding sulla base di alcune proprietà fondamentali, che risultano essenziali per il contesto applicativo:

- **Gestione della polisemia**: Modelli statici come Word2Vec [35] o GloVe [41] non sono in grado di gestire la polisemia, in quanto ogni parola viene rappresentata da un solo vettore, indipendentemente dal contesto. Questo comporta, ad esempio, che parole come *burro* vengano rappresentate in modo simile sia che ci si riferisca all'alimento, sia che si intenda il prodotto cosmetico *burro di karité*, portando a evidenti problemi semantici nel task.

- **Comprensione del contesto**: TF-IDF rappresenta i termini in modo bag-of-words, senza tenere conto dell'ordine o del contesto sintattico e semantico in cui essi compaiono. Sebbene tecniche come FastText [7] riescano a modellare morfologia e sottoparti delle parole, anche esse non risolvono pienamente la polisemia.

- **Supporto multilingua e per l'italiano**: Modelli basati su architetture *Transformer*, come BERT [14] e le sue varianti sentence-level come SBERT [42], risultano essere più adatti a catturare le sfumature semantiche delle keyword anche in italiano, grazie a un'addestramento contestuale su corpus multilingua.

In aggiunta a questi vantaggi semantici, SBERT offre anche benefici computazionali rilevanti. Esistono versioni leggere del modello (es. `all-MiniLM-L6-v2`) che utilizzano solo pochi milioni di parametri (il modello usato 22.7 milioni [26]), mantenendo comunque una buona accuratezza semantica e un'efficienza di calcolo sufficiente per l'embedding di keyword corte e isolate. A differenza di modelli molto più pesanti (come GPT-3, LLaMA o PaLM, con decine di miliardi di parametri), SBERT rappresenta un compromesso ottimale tra prestazioni e costi computazionali per il task considerato.

Pertanto, alla luce di queste considerazioni, è stata scelta l'architettura SBERT come metodo di embedding delle keyword in questo progetto.

### SBERT all-MiniLM-L6-v2: funzionamento e risultati

Il modello `all-MiniLM-L6-v2` è una variante compatta ed efficiente del modello *Sentence-BERT* (SBERT), sviluppata e distribuita tramite la piattaforma *SentenceTransformers* [42]. Questo modello nasce dall'adattamento del modello *MiniLM* [52], che si caratterizza per una struttura Transformer estremamente leggera ma allo stesso tempo molto efficace, con sole 6 layer di encoder.

L'obiettivo principale del modello `all-MiniLM-L6-v2` è generare embedding semantici di frasi e testi di lunghezza arbitraria, in modo che frasi con significato simile siano mappate in rappresentazioni vettoriali (tipicamente nello spazio $\mathbb{R}^{384}$) che risultino vicine tra loro secondo una misura di distanza, ad esempio la *cosine similarity*.

A differenza dei classici modelli di embedding statico come *Word2Vec* [35] o *GloVe* [41], SBERT genera rappresentazioni contestualizzate. Questo significa che due parole identiche inserite in frasi differenti avranno rappresentazioni differenti, in base al contesto sintattico e semantico circostante. Questo comportamento è essenziale per la gestione della polisemia e delle sfumature linguistiche.

**Architettura** Il modello è composto da:

- Un encoder basato su MiniLM, che processa le sequenze di input e ne estrae la rappresentazione contestuale.
- Un pooling layer, che aggrega i vettori token-level in un singolo vettore frase. Tipicamente, la tecnica di *mean pooling* viene utilizzata, ovvero si calcola la media dei vettori token corrispondenti agli elementi della frase.
- Un output finale, che restituisce un vettore denso di dimensione 384 (dimensionalità fissa), utile per task di similarità semantica, clustering o retrieval.

**Output del modello** Il modello prende in input una o più frasi o termini e restituisce un tensore di dimensione $(n, 384)$, dove $n$ è il numero di frasi in input. Ogni vettore può essere interpretato come una rappresentazione semantica della frase corrispondente, che può essere comparata ad altre tramite misure come la similarità coseno:

$$\text{sim}(x, y) = \frac{x \cdot y}{\|x\|\|y\|}$$

Questa rappresentazione è particolarmente adatta a compiti non supervisionati come il clustering semantico (es. con UMAP-HDBSCAN) o la costruzione di motori di ricerca semantici.

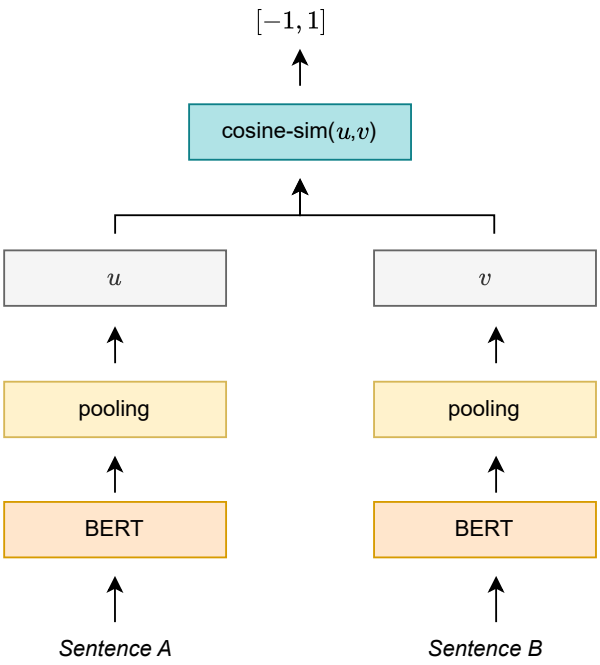


**Figura 2.3:** Una rappresentazione dell'architettura SBERT durante una fase di comparazione tra due sequenze [42]

### Vantaggi

- Alta efficienza computazionale: adatto anche a contesti a bassa latenza e risorse limitate.
- Prestazioni elevate in task di similarità testuale rispetto a TF-IDF e Word2Vec.
- Supporto multilingua e gestione efficace della polisemia.

Il modello `all-MiniLM-L6-v2` è stato addestrato su un'ampia varietà di dataset (tra cui SNLI, MultiNLI, STS benchmark, NLI datasets e paraphrase datasets) e ottimizzato per restituire embedding universalmente utili in molteplici contesti NLP, come classificazione, clustering, retrieval, e semantic search [42].

### 2.2.5 Gestione delle campagne automatiche e delle anomalie nei dati

Durante la fase di pulizia dei dati, successiva alla configurazione dei termini di ricerca, è stato possibile individuare alcune righe anomale nel dataset, in particolare nella colonna *Targeting* e nella colonna *Customer Search Term*. In alcuni casi, la colonna *Targeting* conteneva un codice identificativo prodotto (ASIN) con estensione `asin-expanded`, mentre la colonna *Customer Search Term* riportava o lo stesso codice ASIN oppure un ASIN differente.

Questi casi sono attribuibili a campagne che operano all'interno delle pagine prodotto dei competitor. Se tali campagne risultano configurabili (ovvero è possibile selezionare manualmente gli ASIN target per l'advertising), allora possono essere considerate utili e mantenute nel dataset. In caso contrario, se non vi è possibilità di controllo sull'ASIN, le relative righe vengono eliminate in quanto non coerenti con l'obiettivo di modellazione non supervisionata.

Un secondo gruppo di righe anomale è caratterizzato dalla presenza del carattere `*` nella colonna *Targeting*, mentre nella colonna *Customer Search Term* sono presenti stringhe testuali coerenti con ricerche reali degli utenti, come ad esempio *stuccante da viaggio*, *struccante oleoso*, *burro detergente viso e occhi*, ecc. Tali casi sono riconducibili a campagne automatiche gestite direttamente da Amazon. In queste, l'algoritmo seleziona in autonomia le parole chiave più opportune per mostrare l'annuncio pubblicitario.

Nonostante l'impossibilità di controllare direttamente la selezione delle keyword da parte di Amazon, i termini di ricerca utilizzati dai clienti rappresentano una fonte informativa preziosa. Infatti, possono essere riutilizzati come potenziali keyword target in campagne manuali. A tal fine, si propone di riportare il contenuto della colonna *Customer Search Term* all'interno della colonna *Targeting*, qualora esso non sia un codice ASIN. Viene inoltre assegnata una tipologia di corrispondenza sulla base della specificità della keyword:

- **Corrispondenza ampia (codice 0)**: indica che la keyword può essere utilizzata in modo flessibile, eventualmente integrando parole affini. Ad esempio, dalla keyword *struccante oleoso* si possono derivare varianti come *struccante oleoso burro*.
- **Corrispondenza a frase (codice 1)**: mantiene l'ordine delle parole nella frase, ma consente l'aggiunta di termini prima o dopo.
- **Corrispondenza esatta (codice 2)**: rappresenta l'utilizzo preciso della keyword così come è stata digitata dall'utente, ad esempio *struccante 100 ml*.

È stata esclusa l'ipotesi di indicare nel modello se una campagna è automatica o manuale. Tale informazione, infatti, non risulta rilevante ai fini del processo predittivo, in quanto ciò che realmente interessa al modello sono i risultati in termini di click, costi e conversioni, non la modalità di generazione della campagna.

Infine, si è scelto di filtrare tutte le righe la cui colonna *Targeting* inizia con la stringa `asin-expanded` oppure con il simbolo `*`, secondo la logica descritta sopra. I termini di ricerca rilevanti vengono quindi riposizionati nella colonna *Targeting*.

**Rimozione delle variabili target e prevenzione del data leakage** Durante l'ultima fase di pulizia del dataset, è stato necessario procedere alla rimozione delle variabili che si intendono predire tramite il modello, ovvero: `Impressions`, `Clicks` e `Spend`. Tali colonne rappresentano le variabili target per il task di regressione proposto, pertanto non devono essere incluse nel set di input del modello predittivo.

In aggiunta, anche le metriche derivate da queste variabili, come ad esempio `7 Day Total Sales`, `ACOS` (Advertising Cost of Sales), `ROAS` (Return on Advertising Spend) e `Conversion Rate`, devono essere rimosse dal dataset di training. Tali metriche, infatti, vengono calcolate direttamente a partire dalle variabili target e introdurle tra le feature costituirebbe un chiaro caso di *data leakage*, ovvero l'inclusione di informazioni future o derivate dal target stesso tra le variabili esplicative ([28, 9]). Ciò violerebbe il principio

di indipendenza tra input e target e porterebbe a una sovrastima artificiale della capacità predittiva del modello.

Eventualmente, in lavori futuri, sarà possibile considerare l'introduzione di versioni *laggate* di queste metriche derivate, calcolate su finestre temporali antecedenti rispetto al periodo di previsione. Ad esempio, si potrebbe considerare l'`ACOS` medio della settimana precedente o il `Total Sales` nello stesso mese dell'anno precedente, qualora tali dati storici siano disponibili. Questo approccio consentirebbe di introdurre informazione storica utile senza incorrere nel rischio di data leakage, sfruttando la dipendenza temporale che caratterizza i fenomeni osservati.

> *Nota: attualmente non sono disponibili dati storici sufficienti per implementare strategie di lag temporale avanzate, ma ciò rappresenta un'importante direzione di sviluppo futuro.*

### 2.2.6 Proposta di Ottimizzazione delle Campagne Pubblicitarie Automatiche di Amazon

Nelle campagne pubblicitarie automatiche, Amazon propone prodotti ai clienti in base alle loro ricerche, generando report in cui le campagne sono etichettate con un attributo di targeting, come "loose-match", "phrase-match" o "exact-match", e un campo "customer search term" che riporta la keyword ricercata. Dal punto di vista tecnico, questo dato rappresenta una sfida significativa, poiché il 69% del dataset è composto da ricerche di questo tipo. Pertanto, è necessario sviluppare una soluzione per gestire tali dati in modo efficace.

Partendo dall'assunzione che Amazon utilizza un algoritmo automatico per associare le ricerche degli utenti ai prodotti, si può ipotizzare che il tipo di match ("loose-match", "phrase-match" o "exact-match") indichi il grado di similarità tra la keyword ricercata e il prodotto. Poiché Amazon confronta le ricerche con la pagina prodotto e le parole che la compongono, è ragionevole supporre che venga generata una rappresentazione del tipo "bag of words" per le parole presenti nella pagina prodotto. Tuttavia, non avendo accesso diretto alle parole generate da Amazon, si propone di utilizzare le campagne pubblicitarie manuali per descrivere i prodotti.

Per raggiungere questo obiettivo, si procede unendo il dataset di targeting, contenente le keyword selezionate manualmente, con il dataset dei prodotti pubblicizzati, ottenendo così un'associazione tra keyword e prodotto. Questo passaggio consente di identificare un insieme di keyword che descrivono il prodotto, considerate come una "ground truth" poiché definite manualmente. Successivamente, si definisce il dominio semantico di un prodotto come l'insieme di tutte le sue keyword di targeting. Questo dominio viene confrontato con il "customer search term" per verificare se la ricerca rientra nel dominio del prodotto. Il dataset risultante da questa unione viene denominato "targeting_asin".

Una volta identificato il dominio di appartenenza del "customer search term", si confronta quest'ultimo con tutte le keyword che definiscono il prodotto per individuare la parola più simile. La keyword con la maggiore similarità viene quindi utilizzata come parola di targeting per il "customer search term".

#### Procedimento Tecnico

Per implementare questa strategia, si propone l'utilizzo di un modello di Sentence Transformers, come "all-MiniLM-L6-v2", per generare embedding (vettori numerici) per ogni

| Targeting | Tipo di corrispondenza | Termine ricerca cliente |
|---|---|---|
| struccante | PHRASE | struccanti dischetti |
| burro struccante | EXACT | burro struccante |
| burro struccante | EXACT | burro struccante |
| burro struccante | EXACT | burro struccante |
| burro struccante | EXACT | burro struccante |
| burro struccante | EXACT | burro struccante |
| burro struccante | EXACT | burro struccante |
| burro struccante | EXACT | burro struccante |
| balsamo struccante | EXACT | balsamo struccante |
| balsamo struccante | EXACT | balsamo struccante |
| detergente viso da viaggio | EXACT | detergente viso da viaggio |
| detergente viso da viaggio | EXACT | detergente viso da viaggio |
| detergente viso da viaggio | EXACT | detergente viso da viaggio |
| detergente viso da viaggio | EXACT | detergente viso da viaggio |
| detergente viso da viaggio | EXACT | detergente viso viaggio |
| makeup remover | EXACT | makeup remover |
| makeup remover | EXACT | makeup remover |
| makeup remover | EXACT | makeup remover |
| makeup remover | EXACT | makeup remover |
| makeup remover | EXACT | makeup remover |
| makeup remover | EXACT | makeup remover |
| detergente viso | PHRASE | detergente viso stick |
| detergente viso | PHRASE | weleda detergente viso |
| struccante occhi | EXACT | struccante occhi |
| struccante occhi | EXACT | struccante occhi |
| asin-expanded="B0DKTCNCTB" | - | b0blz5wq8t |
| asin-expanded="B0DKPJ8P6G" | - | b0dqq6vm5b |
| asin-expanded="B0DKTCYD5X" | - | b09cdhb88b |
| asin-expanded="B0DKTCYD5X" | - | b0f3c5jfnq |
| loose-match | - | acido ialuronico 3 pesi molecolari |
| loose-match | - | acido ialuronico viso |
| substitutes | - | acido ialuronico viso |
| loose-match | - | acido ialuronico viso uomo |
| loose-match | - | acido mandelico peeling |
| loose-match | - | acido mandelico viso |
| loose-match | - | allbeauty |

**Figura 2.4:** Una rappresentazione del dataset "Customer search term"

keyword di targeting e ogni "customer search term". Il dominio semantico di un prodotto è rappresentato dall'insieme degli embedding delle sue keyword di targeting. Per verificare l'appartenenza di un "customer search term" al dominio di un prodotto, si calcola la similarità coseno tra l'embedding del "customer search term" e gli embedding delle keyword di targeting del prodotto. La keyword con la similarità più alta determina il grado di appartenenza.

A partire dal dataset ripulito, contenente solo le colonne "ASIN" e "Targeting", si calcolano i vettori SBERT per tutte le keyword di targeting. Il dataset risultante, denominato "dic_targeting", viene salvato per consentirne il riutilizzo. Successivamente, si definisce una funzione che, dato un "customer search term" in input, restituisce l'ASIN e la keyword di targeting più simili. Il procedimento della funzione è il seguente:

1. Calcolare la similarità coseno tra l'embedding del "customer search term" e tutti gli embedding delle keyword nel dataset "dic_targeting".

2. Calcolare la media della similarità per ogni ASIN presente in "dic_targeting".

3. Selezionare l'ASIN con la media di similarità più alta come quello associato al "customer search term".

4. Tra le keyword di targeting associate all'ASIN selezionato, scegliere quella con la maggiore similarità rispetto al "customer search term".

**Punti di Forza**

- **Accuratezza semantica**: L'uso di SBERT per generare gli embedding consente di catturare il significato semantico delle keyword di targeting e dei "customer search term", riconoscendo sinonimi e contesti simili (es. "siero ciglia" vs. "allungatore ciglia").

- **Raggruppamento per ASIN**: Calcolare la media della similarità coseno per ASIN sintetizza il dominio semantico di un prodotto, rendendo la selezione dell'ASIN più robusta rispetto al confronto con una singola keyword.

- **Riutilizzabilità**: Il salvataggio del dataset "dic_targeting" con gli embedding precalcolati riduce il costo computazionale per query future, rendendo il sistema efficiente per utilizzi ripetuti.

- **Output dettagliato**: Restituire sia l'ASIN più simile che la keyword di targeting più vicina fornisce informazioni utili per analisi successive, come l'ottimizzazione delle campagne pubblicitarie.

- **Flessibilità**: La tecnica può essere adattata per includere soglie di similarità o per gestire più ASIN in output.

**Punti di Debolezza**

- **Perdita di dettaglio con la media**: Il calcolo della media della similarità per ASIN può mascherare keyword altamente simili se altre keyword dello stesso ASIN presentano bassa similarità, portando a una scelta meno precisa.

- **Costo computazionale iniziale**: La generazione degli embedding SBERT per tutte le keyword richiede tempo e risorse, soprattutto per dataset di grandi dimensioni, sebbene questa operazione venga eseguita una sola volta.

- **Confronti lineari**: Confrontare il "customer search term" con tutte le keyword di "dic_targeting" è computazionalmente costoso (complessità O(n) per n keyword) senza ottimizzazioni come FAISS.

- **Dipendenza dalla qualità delle keyword**: Se le keyword di targeting per un ASIN sono incomplete o rumorose, il dominio semantico potrebbe non rappresentare accuratamente il prodotto.

- **Soglia implicita**: L'assenza di una soglia di similarità per accettare o rifiutare una corrispondenza rischia di assegnare un "customer search term" a un ASIN anche in presenza di una bassa similarità.

- **Sensibilità agli outlier**: La media della similarità può essere influenzata da keyword outlier con bassa similarità, riducendo l'accuratezza per ASIN con molte keyword eterogenee.

I punti di debolezza descritti non rappresentano, al momento, un ostacolo significativo e possono essere temporaneamente trascurati.

### Analisi dei Risultati e Problemi Rilevati

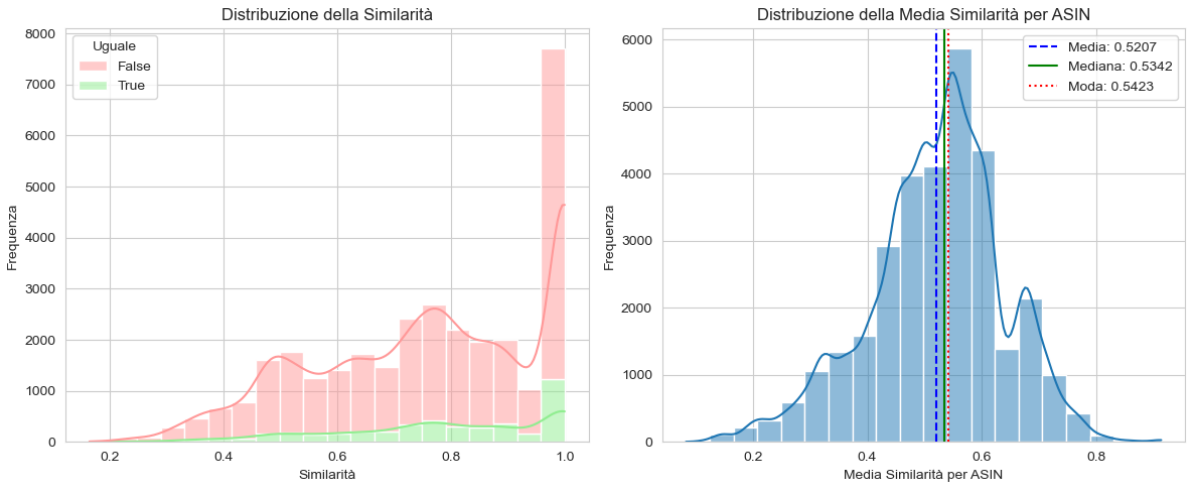


**Figura 2.5:** Distribuzione delle similarità (grafico a sinistra) prima della previsione con sbert (in rosso, *False*) e dopo in verde (*True*). A destra invece il grafico della distribuzione media della similarità per ASIN."

Applicando la tecnica proposta al dataset, si è osservato che la maggior parte delle etichette assegnate durante il merge dei dataset "targeting" e "prodotto pubblicizzato" risulta errata (come si può vedere in fig. 2.5). Questo problema deriva dall'utilizzo di un merge di tipo "left", che comporta la duplicazione delle righe del dataset "customer search term" per ogni ASIN associato a una campagna pubblicitaria con più ASIN. Di conseguenza, se una campagna include, ad esempio, cinque ASIN, vengono generate cinque righe, ma solo una di esse è associata correttamente all'ASIN corrispondente. Questo processo crea numerosi record aggiuntivi che, sebbene utili per l'addestramento di un modello, introducono errori se l'ASIN assegnato è sbagliato, compromettendo la qualità dei dati di addestramento.

Per risolvere questo problema, si propone di utilizzare un modello SBERT per predire l'ASIN, fornendo informazioni più accurate. I risultati ottenuti sono illustrati nel grafico di sinistra, che mostra la similarità tra le keyword del dataset di targeting, considerate come una "bag of words" per descrivere un prodotto, e il "customer search term". Nel grafico, i record errati derivanti dal merge sono evidenziati in rosso, mentre quelli corretti, risultanti dall'unione dei dataset, sono evidenziati in verde. Poiché Amazon non fornisce la "ground truth" per l'associazione tra record e prodotti, si adotta un ragionamento induttivo. Questo approccio non garantisce una conclusione certa, ma mira a dimostrare una ragionevole probabilità di correttezza.

Dal grafico di sinistra si osserva una similarità semantica e grammaticale, garantita da SBERT, tra il "customer search term" e le keyword di targeting che definiscono il prodotto. Tuttavia, l'unione dei dataset assegna spesso un ASIN errato, un fenomeno

che si ripete, sebbene in modo meno marcato, nel resto del grafico. Questo suggerisce che, quando l'ASIN predetto tramite similarità differisce da quello derivante dall'unione dei dataset, il primo risulta generalmente più accurato. Un'analisi manuale dei dati conferma ulteriormente questa tendenza.

**Conclusioni e Proposta Operativa**

Sulla base di queste osservazioni, si decide di utilizzare i risultati forniti dalla funzione "find_closest_asin_and_targeting", che calcola la similarità coseno per determinare l'ASIN e la keyword di targeting più appropriati. Prima di procedere, si è verificata la correttezza del dataset "targeting_asin", che è stato generato con lo stesso metodo di unione utilizzato per il dataset principale. Data la dimensione ridotta di "targeting_asin", è stato possibile effettuare una supervisione manuale per garantirne l'accuratezza.

### 2.2.7 Analisi Esplorativa dei Dati (EDA)

Di seguito vengono descritte le tecniche di visualizzazione utilizzate per l'analisi esplorativa del dataset relativo alle campagne pubblicitarie su Amazon. Ogni tecnica è accompagnata da una spiegazione del suo scopo e delle feature considerate, con l'obiettivo di evidenziare pattern, relazioni e anomalie nei dati.

## Istogrammi delle Variabili Numeriche

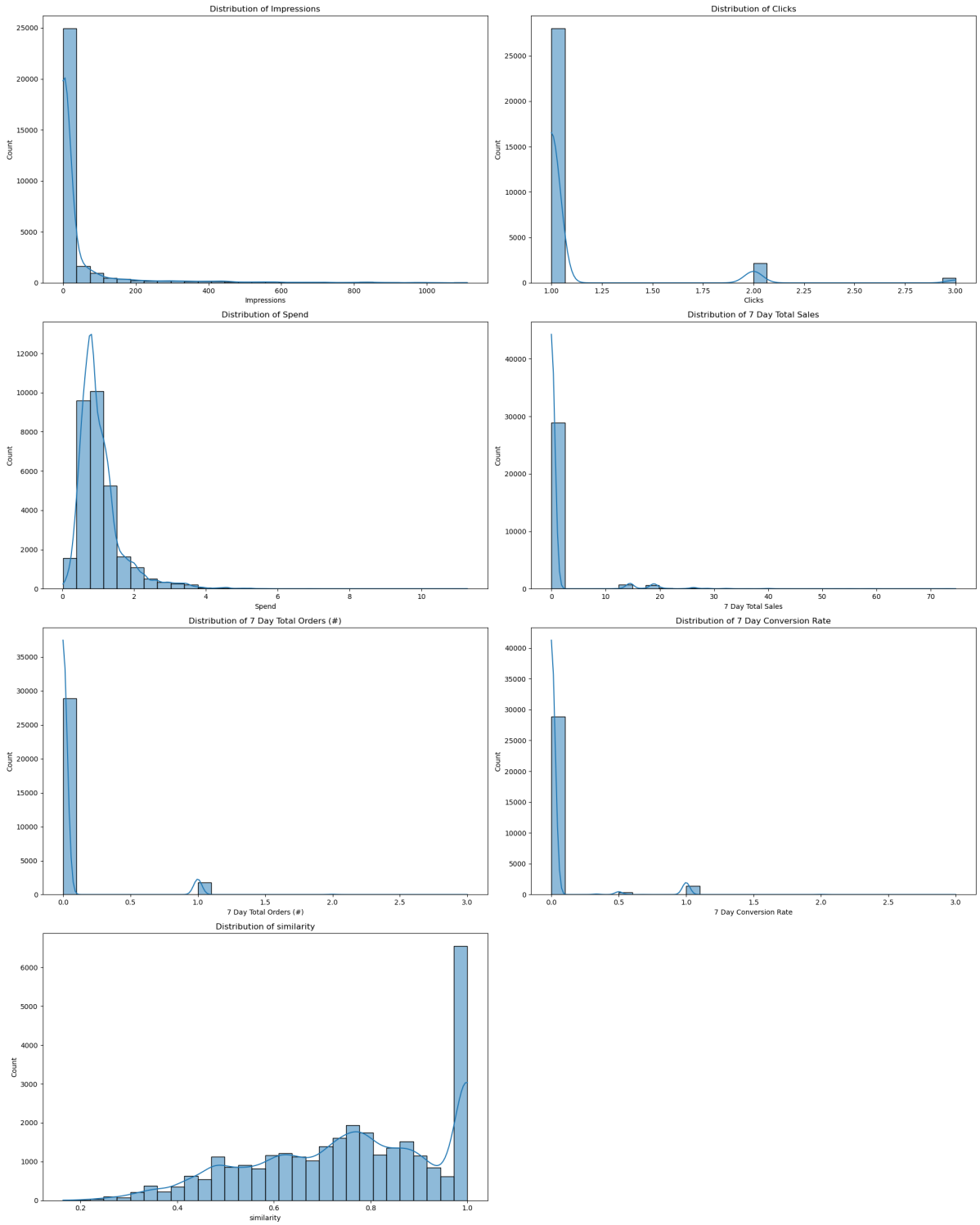


**Figura 2.6:** Istogramma delle feature numeriche

Gli istogrammi mostrano la distribuzione delle variabili numeriche, permettendo di analizzare la forma della distribuzione, la presenza di asimmetrie e outlier. Questa tecnica è utile per comprendere il comportamento delle metriche chiave delle campagne pubblicitarie.

**Feature considerate**: `Impressions`, `Clicks`, `Spend`, `7 Day Total Sales`, `7 Day Total Orders (#)`, `7 Day Conversion Rate`, `similarity`.

**Scopo**: Identificare distribuzioni skewed, outlier e necessità di trasformazioni per successive analisi statistiche o di modellazione.

### Grafici a Barre delle Variabili Categoriche

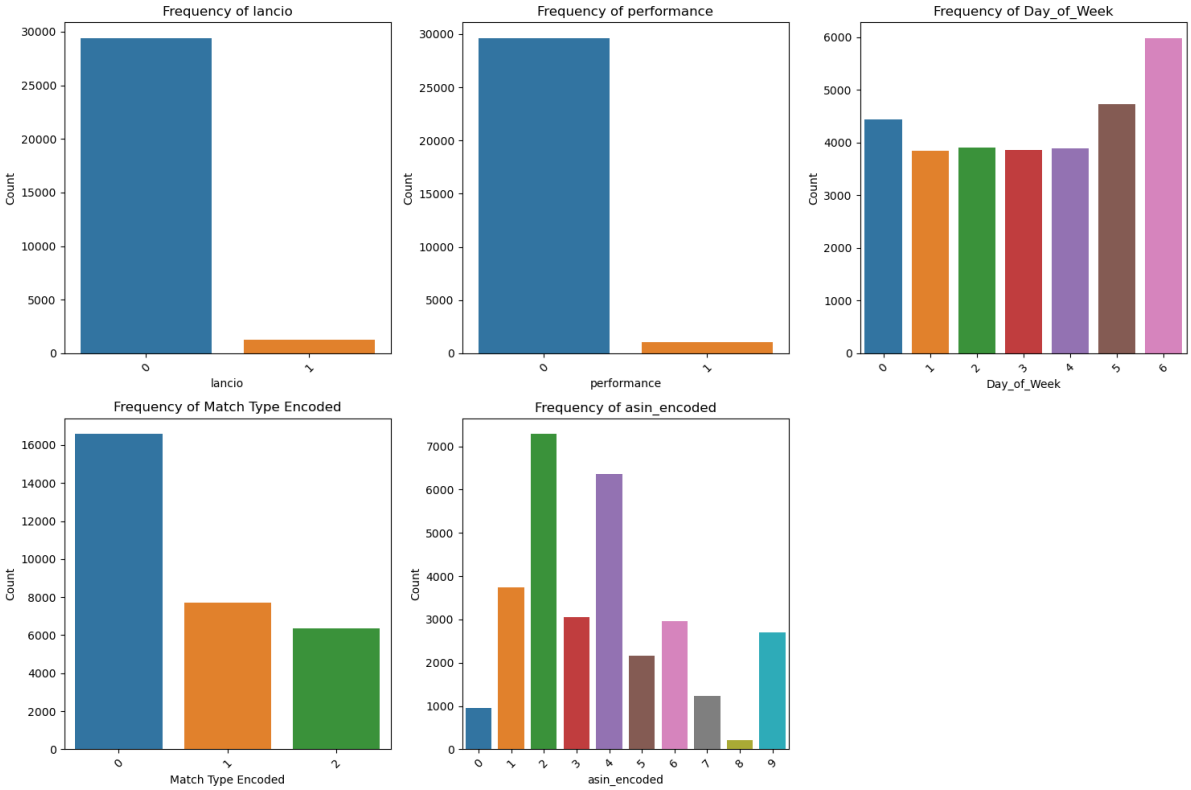


**Figura 2.7:** Grafici a Barre delle Variabili Categoriche

**Considerazioni** : I grafici a barre visualizzano la frequenza delle categorie per le variabili categoriche, evidenziando la distribuzione dei dati e l'eventuale squilibrio tra le classi.

**Feature considerate**: `lancio`, `performance`, `Day_of_Week`, `Match Type Encoded`, `asin_encoded`.

**Scopo**: Valutare la rappresentatività delle categorie (ad esempio, bilanciamento di `lancio`) e identificare categorie dominanti (ad esempio, ASIN più frequenti).

### Heatmap delle Correlazioni

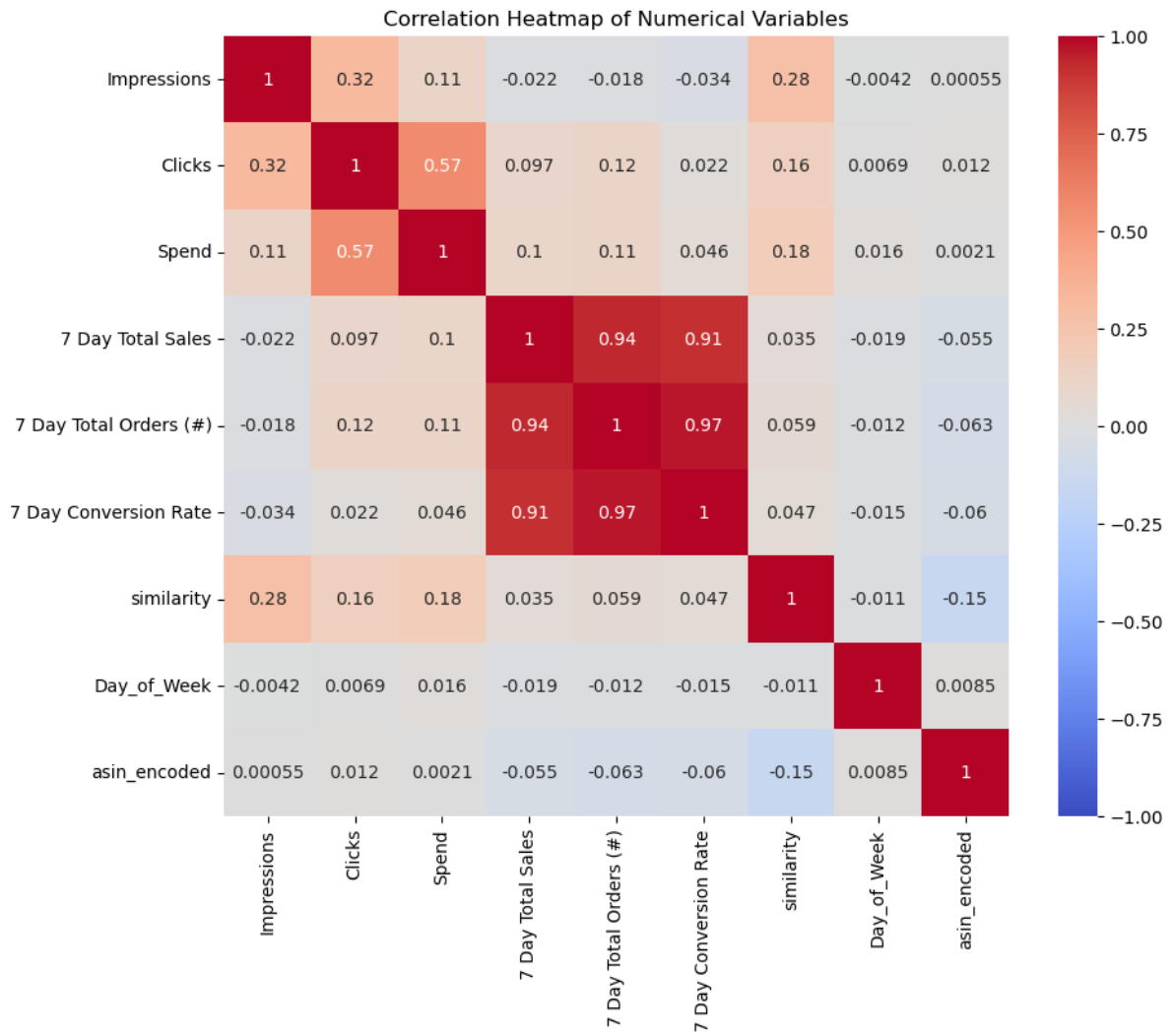


**Figura 2.8:** Correlation heatmap

La heatmap (fig. 2.8) visualizza la matrice di correlazione tra variabili numeriche, evidenziando relazioni lineari forti o deboli attraverso coefficienti di correlazione.

**Feature considerate**: `Impressions`, `Clicks`, `Spend`, `7 Day Total Sales`, `7 Day Total Orders (#)`, `7 Day Conversion Rate`, `similarity`.

**Scopo**: Identificare multicollinearità (ad esempio, tra `Clicks` e `Impressions`) e variabili con bassa correlazione con `7 Day Total Sales`, utili per la selezione delle feature in modelli predittivi.

### Considerazioni

**Considerazioni** Dall'analisi della *correlation heatmap* emergono diverse osservazioni rilevanti. In primo luogo, si nota una fortissima correlazione positiva tra le variabili `7 Day Total Sales`, `7 Day Total Orders (#)` e `7 Day Conversion Rate`, con coefficienti di correlazione compresi tra 0,91 e 0,97. Questo comportamento è atteso, poiché un aumento del numero di ordini e del tasso di conversione si riflette direttamente sull'incremento delle vendite.

La variabile `Spend` presenta una moderata correlazione positiva con `Clicks` (0,57), indicando che una maggiore spesa pubblicitaria tende ad aumentare il numero di clic ricevuti. La stessa `Spend` mostra una debole correlazione con le vendite e gli ordini settimanali (valori tra 0,10 e 0,11), suggerendo che una spesa maggiore non garantisce necessariamente un incremento proporzionale nelle vendite o negli ordini. Questo evidenzia possibili inefficienze nelle strategie di allocazione del budget.

La metrica `similarity`, mostra una correlazione debole con tutte le altre variabili. In particolare, il valore massimo è con `Spend` (0,18), mentre le correlazioni con `7 Day Total Sales`, `7 Day Orders (#)` e `7 Day Conversion Rate` sono molto basse (rispettivamente 0,035, 0,059 e 0,047). Questo indica che, nel dataset analizzato, la similarità semantica tra keyword e query non influisce significativamente sulle performance di vendita, lasciando intendere che altri fattori potrebbero avere un ruolo più determinante. Tuttavia, la correlazione tra le feature `similarity`, `impression` mostra come ci sia una correlazione che definisce il concetto di "pertinenza di ricerca" cioè un utente che trova un prodotto che è più correlato con la sua ricerca è più probabile che acquisti rispetto a quando il prodotto viene proposto dopo una ricerca generica.

Infine, le variabili `Day_of_Week` e `asin_encoded` non mostrano correlazioni rilevanti con le metriche di performance, confermando la loro natura più categoriale e indicativa di effetti non lineari o non direttamente correlabili.

Nel complesso, l'analisi suggerisce che l'efficacia di una campagna non può essere spiegata da una singola variabile, ma risulta dall'interazione di molteplici fattori. Le metriche di conversione e ordini settimanali sono fortemente collegate, mentre la spesa e la similarità semantica mostrano solo una parziale relazione con le performance, indicando potenziali aree di ottimizzazione.

### Matrice di Scatter Plot (Pair Plot)


**Figura 2.9:** Analisi bivariata: Matrice di Scatter Plot (Pair Plot)

La matrice di scatter plot esplora le relazioni bivariate tra variabili numeriche, con colorazione basata su una variabile categorica. Consente di individuare correlazioni, pattern non lineari e cluster.

**Feature considerate**: `Impressions`, `Clicks`, `Spend`, `7 Day Total Sales`, `7 Day Total Orders (#)`, `7 Day Conversion Rate`, `similarity`, con colorazione per `lancio`.

**Scopo**: Analizzare le relazioni tra metriche (ad esempio, `Clicks` vs. `Impressions`) e verificare se le campagne di lancio (`lancio=1`) si distinguono in termini di prestazioni.

**Considerazioni** La *scatter plot matrix* rappresenta un'analisi bivariata delle principali variabili numeriche del dataset, stratificata per tipologia di match (`PHRASE`, `EXACT`, `BROAD`). In primo luogo, si osserva una distribuzione fortemente asimmetrica per la maggior parte delle variabili, in particolare per `Impressions`, `Clicks` e `Spend`, dove i dati tendono a concentrarsi in prossimità dello zero con poche osservazioni su valori

elevati. Questo fenomeno di *right-skewness* è particolarmente evidente nelle variabili `Impressions` e `Spend`, e suggerisce un forte effetto di polarizzazione nelle performance delle campagne.

Dal punto di vista delle relazioni bivariate, si osserva una tendenza crescente tra `Spend`, `Clicks` e `7 Day Total Sales`, che evidenzia una relazione positiva tra investimento pubblicitario, interazioni e vendite. Tale pattern è ben visibile nei plot `Spend-Clicks` e `Spend-7 Day Total Sales`, con una dispersione coerente tra le tipologie di match, sebbene il tipo `EXACT` sembri leggermente più concentrato su valori più alti di `similarity`.

Un'osservazione interessante emerge dall'interazione tra la variabile `similarity` e le metriche di performance (`7 Day Total Sales`, `7 Day Total Orders (#)` e `7 Day Conversion Rate`). La variabile `similarity`, pur presentando un range continuo tra 0 e 1, mostra una concentrazione notevole di valori intorno a 1 per il tipo di match `EXACT`, confermando che le campagne a corrispondenza esatta tendono ad avere una maggiore coerenza semantica con i termini di ricerca degli utenti. Tuttavia, tale aumento di similarità non si traduce automaticamente in un incremento netto delle vendite o delle conversioni, suggerendo che la rilevanza semantica è solo uno dei fattori che influenzano le performance.

Infine, l'analisi congiunta delle variabili `7 Day Total Orders (#)` e `7 Day Conversion Rate` conferma quanto già evidenziato nella heatmap: esiste una forte correlazione tra queste due metriche, visibile nella loro disposizione allineata e concentrata. I dati sembrano suggerire una differenziazione parziale delle performance in base al tipo di match, ma non emergono cluster ben definiti, il che indica che la variabilità intra-classe è elevata e che il tipo di match da solo non è un predittore sufficiente della performance.

Nel complesso, la scatter plot matrix evidenzia relazioni coerenti ma non lineari tra le variabili, nonché l'esistenza di una forte eterogeneità nei dati. La presenza di outlier e distribuzioni asimmetriche suggerisce l'opportunità di utilizzare tecniche di normalizzazione e metodi robusti nella fase di modellazione.

### Box Plot per Variabili Categoriche

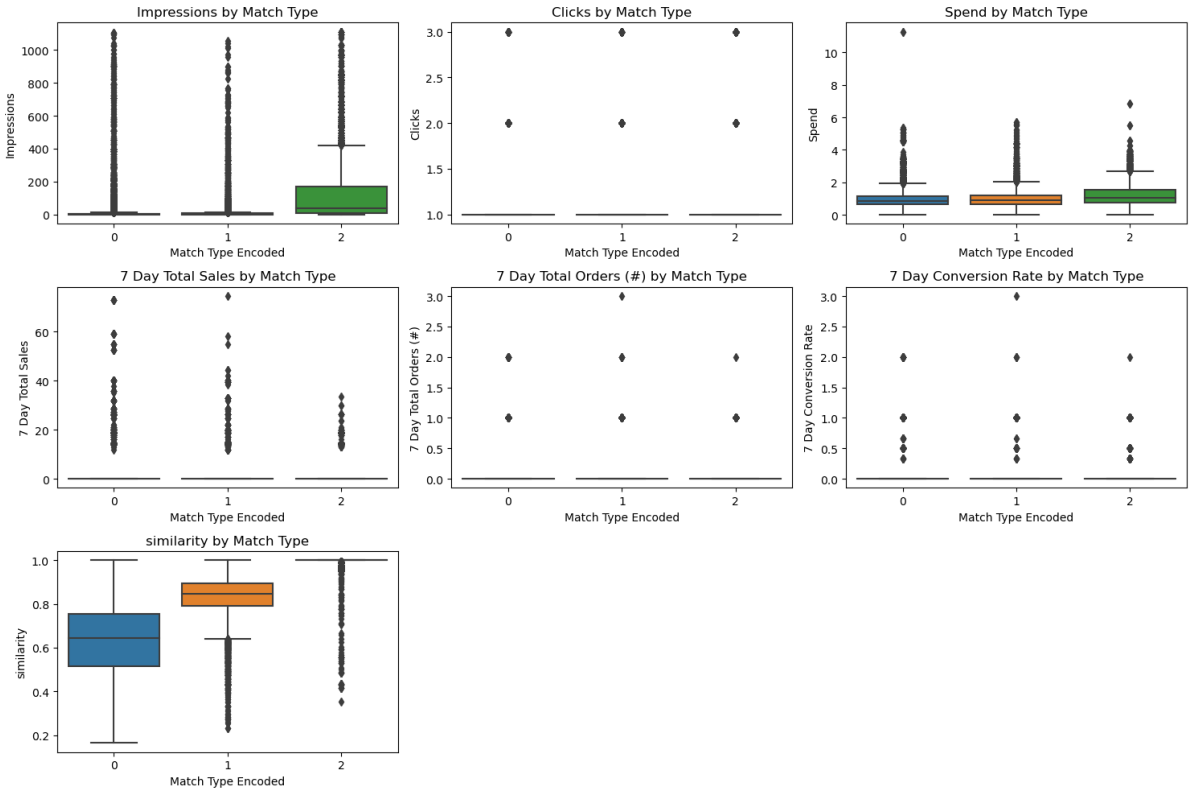


**Figura 2.10:** Box Plot per Variabili Categoriche

I box plot confrontano la distribuzione delle variabili numeriche rispetto alle categorie di una variabile categorica, evidenziando differenze nei valori centrali, variabilità e outlier.

**Feature considerate**: `Impressions`, `Clicks`, `Spend`, `7 Day Total Sales`, `7 Day Total Orders (#)`, `7 Day Conversion Rate` rispetto a `Match Type Encoded`, `lancio`, `performance`, `Day_of_Week`.

**Scopo**: Valutare se alcune categorie (ad esempio, `Match Type Encoded=2` per corrispondenza esatta) presentano prestazioni superiori o outlier significativi.

**Considerazioni** Il box plot delle variabili numeriche stratificate per tipo di match (`Match Type Encoded`: 0=`BROAD`, 1=`PHRASE`, 2=`EXACT`) consente di analizzare la distribuzione e la dispersione delle performance delle campagne pubblicitarie in funzione della strategia semantica adottata.

Per quanto riguarda le **Impressions**, si osserva una distribuzione fortemente asimmetrica con la presenza di numerosi outlier per tutti i tipi di match. Tuttavia, il tipo `EXACT` (valore 2) mostra una mediana più alta e una maggiore dispersione, suggerendo che questa configurazione tende ad attivare maggiormente gli annunci.

La variabile `Clicks` presenta una distribuzione estremamente concentrata, con valori prossimi a 1 per la maggior parte dei dati, indipendentemente dal tipo di match. Ciò indica una bassa variabilità nella quantità di clic generati, che potrebbe dipendere da un basso numero complessivo di interazioni o da una limitata efficacia degli annunci.

Nel caso della **spesa pubblicitaria** (`Spend`), i box plot indicano che i tre tipi di match hanno una distribuzione simile, ma `EXACT` mostra una maggiore ampiezza inter-

quartile, con valori massimi più elevati. Questo potrebbe indicare un comportamento più aggressivo in termini di offerta per questa categoria.

Analizzando le metriche di performance diretta, ovvero `7 Day Total Sales`, `7 Day Total Orders (#)` e `7 Day Conversion Rate`, si nota una prevalenza di zeri (assenza di vendite/conversioni), accompagnata da pochi outlier elevati. Questo fenomeno riflette una dinamica di *long-tail* tipica del commercio elettronico, dove poche keyword generano la maggior parte delle vendite. Le tre tipologie di match mostrano comportamenti simili, con una leggera tendenza del tipo `EXACT` a registrare valori massimi leggermente più alti, pur in presenza di elevata varianza.

Infine, la variabile **similarity** evidenzia differenze più marcate. I match `EXACT` mostrano una distribuzione molto concentrata su valori alti (mediana $\sim 0.9$), coerente con la definizione semantica di questa categoria. Al contrario, `BROAD` evidenzia una maggiore variabilità, con una mediana attorno a 0.6, riflettendo una corrispondenza semantica più debole con i termini di ricerca degli utenti. Il tipo `PHRASE` si colloca in posizione intermedia.

In sintesi, i box plot rivelano che, mentre alcune variabili mostrano una distribuzione relativamente simile tra i tipi di match (es. `Spend`), altre come `similarity` e `Impressions` differenziano più nettamente i gruppi, suggerendo che la scelta della tipologia di match può influenzare in modo significativo la copertura e la pertinenza semantica, ma con impatto meno prevedibile sulle performance dirette (vendite e ordini).

### Grafico di dispersione delle Metriche nel Tempo


**Figura 2.11:** Grafico di dispersione delle Metriche nel Tempo

Il grafico a linee mostra l'andamento delle metriche chiave rispetto alla variabile temporale `Date`, permettendo di identificare trend e stagionalità.

**Feature considerate**: `Date`, `Impressions`, `Clicks`, `Spend`, `7 Day Total Sales`.
**Scopo**: Rilevare pattern temporali, come picchi di vendite in periodi specifici, e correlazioni tra spesa e vendite nel tempo.

**Considerazioni** Il grafico mostra l'andamento temporale delle principali metriche di performance nei primi 180 giorni del dataset, dal grafico è stato esclusi il periodo successivo ai 180 giorni per aumentarne la leggibilità dei dati. Le metriche visualizzate includono le `7 Day Total Sales`, i `Clicks`, la `Spesa pubblicitaria (Spend)` e la `7 Day Conversion Rate`, ciascuna riportata tramite una serie di punti nel dominio temporale.

Si osservano chiaramente tre macro-intervalli temporali separati da lacune nei dati (attorno ai giorni 30-75), che indicano interruzioni nella raccolta dati(in quanto non l'azienda non era presente nel marketplace in quelle date). Questo può rappresentare una criticità per la modellizzazione temporale, richiedendo eventualmente tecniche di imputazione o l'esclusione di tali periodi discontinui.

Dal punto di vista delle **vendite settimanali** (`7 Day Total Sales`, asse sinistro), si notano numerosi punti aggregati su livelli costanti (es. 27, 30), probabilmente causati da arrotondamenti o soglie interne di reportistica. La distribuzione è fortemente eterogenea, con alcuni picchi isolati che superano il valore di 70, suggerendo eventi promozionali o anomalie temporanee di alta performance.

La **spesa** (`Spend`) mostra un comportamento più denso e continuo, con una concentrazione significativa attorno a valori compresi tra 1 e 4, e pochi outlier più alti. Questo denota una certa stabilità nei budget pubblicitari giornalieri, ma anche una gestione variabile tra i diversi giorni.

I **clic** (`Clicks`) risultano più uniformi, con una netta concentrazione intorno ai valori 1–2. Questo può suggerire un basso tasso di coinvolgimento generale da parte degli utenti, oppure una limitata esposizione agli annunci.

Infine, la **conversion rate settimanale** (`7 Day Conversion Rate`) è generalmente bassa, spesso prossima allo zero. Tuttavia, si notano linee orizzontali che indicano valori discretizzati (es. 0.5, 1.0, 1.5), potenzialmente dovute alla definizione metrica su un basso numero di conversioni, con conseguente quantizzazione artificiale della metrica.

In sintesi, il grafico evidenzia una forte variabilità intertemporale e discrepanze tra la spesa sostenuta e i risultati ottenuti, con una tendenza generale a bassa conversione. Le interruzioni nei dati e la discreta densità di valori nulli (soprattutto per le vendite e la conversion rate) suggeriscono che i modelli predittivi debbano essere particolarmente robusti a dati sparsi e a dinamiche discontinue.

### Metriche Medie per Giorno della Settimana

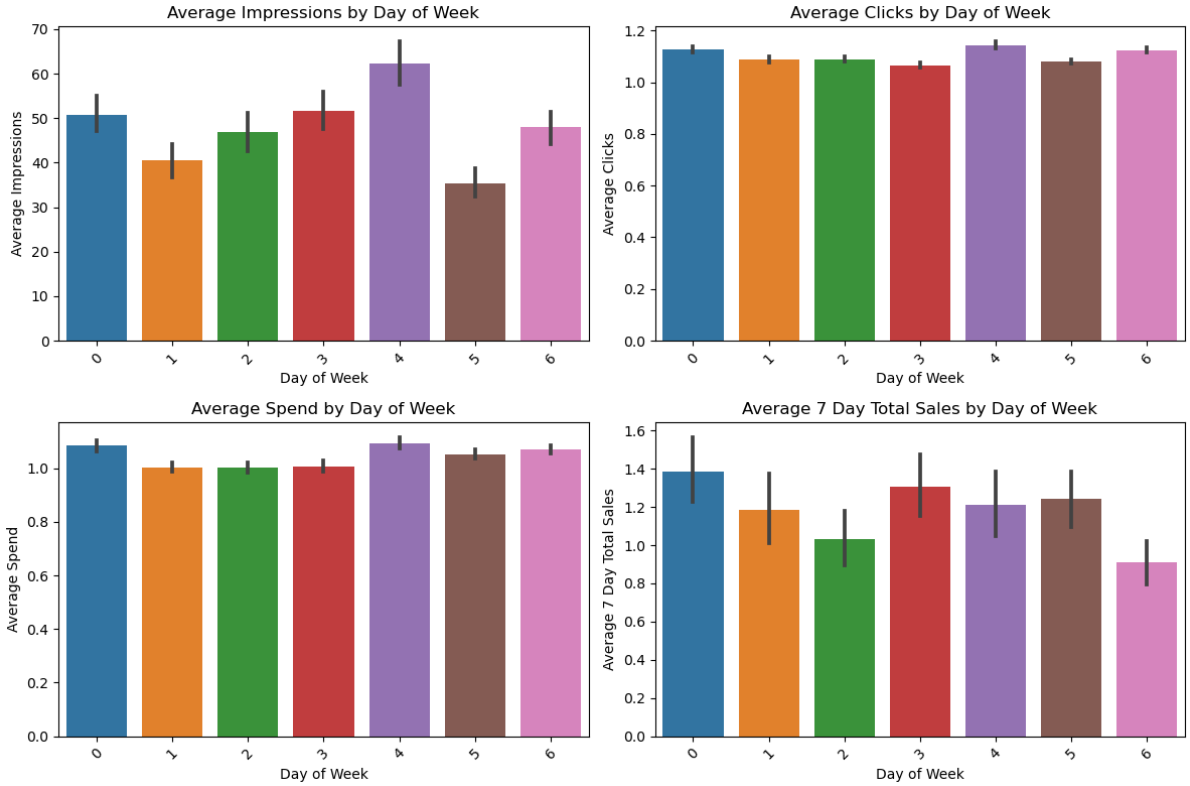


**Figura 2.12:** Metriche Medie per Giorno della Settimana

I grafici a barre mostrano la media delle metriche chiave per ciascun giorno della settimana, evidenziando variazioni giornaliere nelle prestazioni delle campagne.

**Feature considerate**: `Day_of_Week`, `Impressions`, `Clicks`, `Spend`, `7 Day Total Sales`.

**Scopo**: Identificare giorni con prestazioni superiori (ad esempio, fine settimana) per ottimizzare la pianificazione delle campagne.

**Considerazioni** Analizzando il grafico che riporta le medie giornaliere di diverse metriche (Impressioni, Click, Spesa e Vendite totali su 7 giorni), emergono alcune osservazioni significative. Le metriche sono aggregate per **giorno della settimana** (da 0 a 6), corrispondenti a **Lunedì** (0) e **Domenica** (6) (in base a quanto stabilito dall'API pandas per convertire i giorni di una data nei giorni in una settimana [1]).

- **Impressioni medie**: Le impressioni mostrano un picco evidente nel giorno 3 (mercoledì), con un valore approssimativo di 65, mentre i giorni 4 (giovedì) e 5 (venerdì) registrano i valori più bassi, intorno a 35–40. Questo suggerisce una maggiore visibilità delle campagne a metà settimana, potenzialmente legata a una maggiore attività degli utenti in tali giorni.

- **Click medi**: I click raggiungono il valore massimo nei giorni 0 (Lunedì) e 6 (Domenica), con un valore vicino a 1.2, indicando una maggiore interazione degli utenti a fine e inizio settimana. I giorni centrali della settimana (2–4) mostrano valori

inferiori (circa 0.8–1.0), suggerendo un comportamento di engagement più selettivo durante tali giorni.

- **Spesa media**: La spesa media rimane relativamente stabile tra 0.9 e 1.0 per tutti i giorni, con un lieve incremento nei Lunedì e Giovedì. Questo indica una strategia di budget relativamente uniforme, con possibili aggiustamenti per massimizzare l'esposizione (Giovedì) o l'engagement (Lunedì).
- **Vendite totali su 7 giorni**: Le vendite mostrano un picco nel giorno 0 (Lunedì), con un valore di circa 1.4, seguito da un calo nei giorni 1 e 2 (Martedì e Mercoledì, intorno a 1.0–1.1). I giorni 4 e 5 (Venerdì e Sabato) presentano valori più bassi (circa 0.8–0.9), suggerendo che le conversioni siano più probabili nei fine settimana o all'inizio della settimana.

In termini statistici, si osserva una possibile correlazione positiva tra impressioni e click nei giorni di picco (0 e 3), mentre la spesa sembra non correlata con le vendite, indicando che l'allocazione del budget potrebbe non essere ottimizzata per massimizzare le conversioni. Un'analisi più approfondita, ad esempio tramite un modello di regressione lineare, potrebbe quantificare queste relazioni.

Inoltre, la variabilità giornaliera suggerisce la necessità di un'ottimizzazione temporale delle campagne, con un focus su domeniche e mercoledì per massimizzare rispettivamente engagement e visibilità.

### Scatter Plot di Tasso di Conversione vs. Similarità

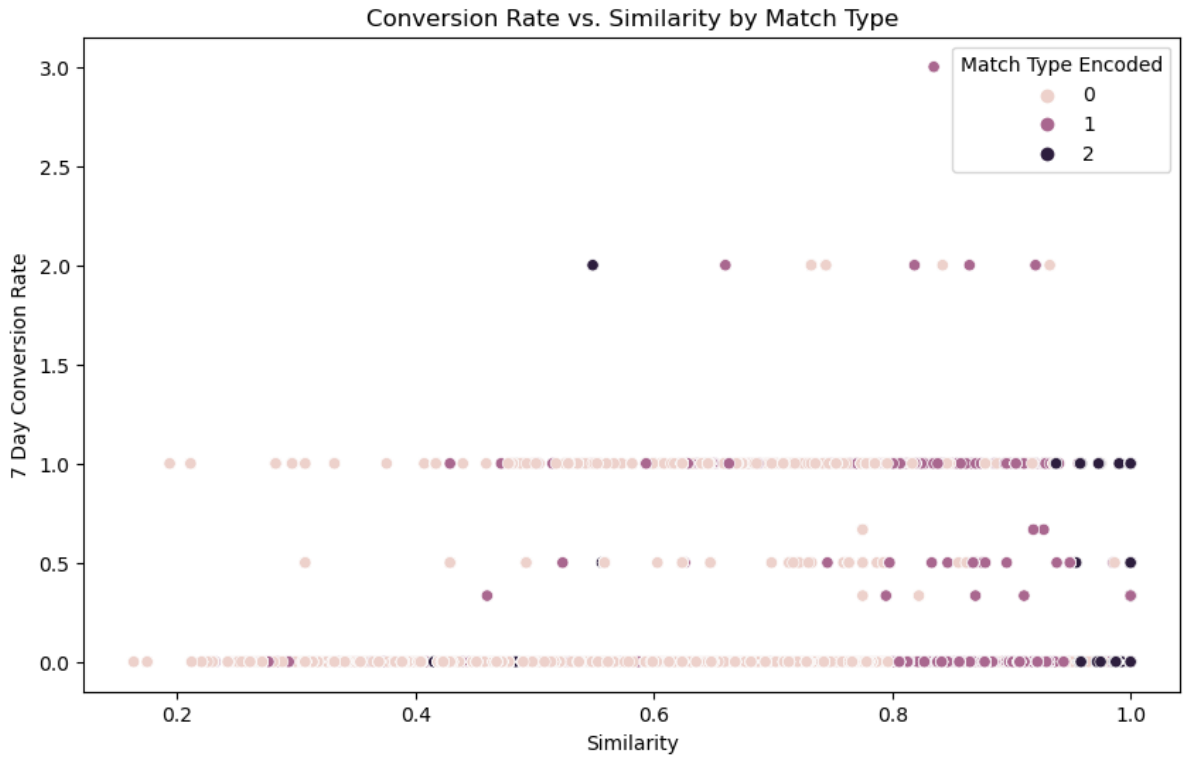


**Figura 2.13:** Scatter Plot di Tasso di Conversione vs. Similarità

Lo scatter plot esplora la relazione tra `similarity` e `7 Day Conversion Rate`, con colorazione per `Match Type Encoded`.

**Feature considerate**: `similarity`, `7 Day Conversion Rate`, `Match Type Encoded`.

**Scopo**: Valutare se una maggiore similarità tra targeting e termini di ricerca dei clienti migliora il tasso di conversione.

**Considerazioni** L'analisi del scatter plot rivela diverse considerazioni:

**Distribuzione eterogenea dei dati:** La maggior parte dei punti dati si concentra nella regione di bassa conversione ($0 \leq$ Conversion Rate $\leq 1$) e alta similarità ($0.7 \leq$ Similarity $\leq 1.0$), suggerendo che le campagne tendono ad essere ottimizzate per alta rilevanza semantica ma non necessariamente per performance di conversione.

**Assenza di correlazione lineare:** Non emerge una correlazione monotonica evidente tra similarità e tasso di conversione. Questo indica che l'alta similarità semantica non è condizione sufficiente né necessaria per garantire elevate performance di conversione, andando in leggero contrasto con l'assunzione che maggiore rilevanza implichi migliori risultati.

**Segmentazione per Match Type:** I diversi match type mostrano pattern distributivi distinti. Il Match Type 2 (EXACT) presenta una concentrazione maggiore in regioni di alta similarità e bassa conversione in quanto c'è una maggiore concentrazione di record con similarity molto alta e a causa del fenomeno long-tail (già riportato nella sezione "Box Plot per Variabili Categoriche") non è possibile osservare un alta concentrazione nei valori di conversion rate maggiori. mentre i Match Type 0 e 1 (BROAD e PHRASE) mostrano una distribuzione più dispersa, suggerendo strategie di targeting differenziate.

**Outliers ad alta conversione:** Si osservano alcuni punti isolati con Conversion Rate $>$ 2.0 distribuiti su diversi livelli di similarità, indicando l'esistenza di configurazioni di campagna altamente performanti che non dipendono esclusivamente dalla similarità semantica.

**Saturazione della similarità:** La concentrazione di punti nella regione Similarity $>$ 0.8 suggerisce un possibile effetto di saturazione, dove incrementi marginali di similarità non producono miglioramenti proporzionali nelle metriche di conversione.

**Implicazioni strategiche:** I risultati suggeriscono la necessità di un approccio multi-dimensionale nell'ottimizzazione delle campagne, dove la similarità semantica rappresenta solo uno dei fattori determinanti.

# Capitolo 3

# Esperimenti effettuati

## 3.1 Obiettivi e Struttura del Capitolo

In questo capitolo verranno mostrati: il percorso metodologico, gli algoritmi valutati, la pipeline implementativa e la motivazione della scelta finale.

Il lavoro di tesi si pone come obiettivo principale la minimizzazione del costo delle campagne pubblicitarie *Pay-Per-Click* (PPC) di Amazon, seguendo criteri che bilancino l'ottimizzazione della spesa con la massimizzazione del ritorno sull'investimento. I dati a disposizione comprendono, per ciascun giorno e per ogni keyword di ogni gruppo di annunci, il numero di click, il numero di impressioni, il numero di acquisti e il numero di vendite, oltre ad ulteriori variabili generate durante la fase di preprocessing. In particolare, sono stati introdotti: gli *embedding* della parola di ricerca del consumatore, la similarità rispetto alla keyword inserita manualmente, il giorno della settimana in cui il record è stato campionato e il tipo di *match* risultante. Tutti questi aspetti sono stati ampiamente trattati nel Capitolo 2, dedicato alla preparazione e analisi preliminare dei dati.

In questo capitolo, invece, l'attenzione si focalizza sui modelli predittivi applicati e sul percorso metodologico che ha condotto alla soluzione finale. La sperimentazione è iniziata con modelli classici di regressione, tra cui `LightGBM`, `XGBoost` e `Random Forest`, utilizzati come baseline. Successivamente, sono state sviluppate e valutate diverse reti *Multi-Layer Perceptron* (MLP), variando la profondità e la complessità architetturale, ed ottimizzando gli iperparametri mediante *grid search*. È stato inoltre sperimentato l'utilizzo di modelli di tipo `LSTM` per la gestione delle sequenze temporali.

Tra le soluzioni analizzate, è stata selezionata una configurazione specifica di MLP come modello di riferimento. Su tale base è stato quindi sviluppato un modello di tipo `contextual bandit`, inizializzato con la MLP precedentemente addestrata, con l'obiettivo di affrontare in maniera dinamica il problema dell'allocazione dei budget pubblicitari attraverso il bilanciamento tra esplorazione e sfruttamento. Per ciascun esperimento condotto sono stati raccolti e salvati i risultati in tabelle `.csv`, al fine di consentire successive analisi e confronti quantitativi.

La struttura del capitolo è dunque organizzata come segue:

- Sezione 3.2: descrizione dei modelli di regressione classici utilizzati come baseline;
- Sezione 3.3: sperimentazione con reti neurali MLP e ottimizzazione degli iperparametri;

- Sezione 3.4: esperimenti con modelli `LSTM` per la modellazione temporale;
- Sezione 3.5: presentazione del modello `contextual bandit` e della sua integrazione con la `MLP`;
- Sezione 3.6: modalità di raccolta, organizzazione e valutazione dei risultati ottenuti.

## 3.2 Modelli di regressione

### 3.2.1 LightGBM

`LightGBM` (Light Gradient Boosting Machine) è un framework di gradient boosting ottimizzato per problemi di regressione e classificazione su dataset strutturati, scelto come uno dei modelli di *baseline* per la sua efficienza computazionale e capacità di gestire dataset ad alta dimensionalità, come quelli descritti nel Capitolo 2. Il modello si basa su un algoritmo di boosting che costruisce alberi decisionali in modo iterativo, minimizzando una funzione di perdita (in questo caso, l'errore quadratico medio, MSE) attraverso una combinazione di gradienti e tecniche di regolarizzazione.

#### Funzionamento del Modello

`LightGBM` utilizza un approccio *histogram-based* per discretizzare le feature continue, riducendo il costo computazionale e migliorando la scalabilità su dataset di grandi dimensioni.[30] Inoltre, supporta feature categoriali native, evitando la necessità di codifiche aggiuntive come il *one-hot encoding*. Nel contesto di questa tesi, `LightGBM` è stato impiegato per prevedere simultaneamente più variabili target (click, spesa, vendite totali a 7 giorni, ordini totali a 7 giorni, tasso di conversione a 7 giorni e impressioni) utilizzando un approccio di regressione multi-output tramite la classe `MultiOutputRegressor` di scikit-learn. Le target sono state trasformate applicando una trasformazione logaritmica (`log1p`) per gestire la loro distribuzione non normale, come descritto nel Capitolo 2. Le predizioni sono state successivamente ritrasformate utilizzando la funzione inversa (`expm1`) per ottenere i valori originali.

#### Implementazione

Il modello è stato implementato in Python utilizzando la libreria `lightgbm` integrata con `scikit-learn`. Le feature di input includono variabili categoriali (`lancio`, `performance`, `Day_of_Week`, `Match Type Encoded`) e numeriche (similarità tra query e *keyword*, e 384 embedding SBERT della parola di ricerca del consumatore, come descritto nel Capitolo 2). Il dataset è stato suddiviso in un set di addestramento (70%) e un set di test (30%) utilizzando uno *shuffle* casuale con `random_state=42` per garantire riproducibilità. Il modello è stato configurato come segue:

- **Obiettivo**: Regressione multi-output con perdita quadratica (`objective='regression'`).
- **Wrapper**: `MultiOutputRegressor` per gestire multiple variabili target.
- **Feature categoriali**: Specificate direttamente nell'addestramento per sfruttare il supporto nativo di `LightGBM`.

Il codice di implementazione è riportato di seguito per chiarezza:

```
from sklearn.multioutput import MultiOutputRegressor
import lightgbm as lgb
import numpy as np
from sklearn.model_selection import train_test_split

# Definisci feature e target
features = ['lancio', 'performance', 'Day_of_Week', 'Match Type
    Encoded',
            'similarity'] + [f'sbert_{i}' for i in range(384)]
targets = ['Clicks', 'Spend', '7 Day Total Sales', '7 Day Total Orders
    (#)',
           '7 Day Conversion Rate', 'Impressions']
X = df[features]
y = df[targets]

# Trasformazione logaritmica delle target
y_transformed = np.log1p(y)

# Split dei dati
X_train, X_test, y_train, y_test = train_test_split(
    X, y_transformed, test_size=0.3, random_state=42, shuffle=True
)

# Configura il modello
model = MultiOutputRegressor(lgb.LGBMRegressor(
    objective='regression', n_estimators=100, learning_rate=0.1
))

# Addestramento
model.fit(
    X_train, y_train,
    categorical_feature=['lancio', 'performance', 'Day_of_Week',
        'Match Type Encoded']
)

# Predizioni
y_test_pred_transformed = model.predict(X_test)
y_test_pred = np.expm1(y_test_pred_transformed)
```

### Iperparametri Testati

Gli iperparametri configurati per il modello `LightGBM` includono:

- `n_estimators=100`: Numero di alberi di boosting.
- `learning_rate=0.1`: Tasso di apprendimento per il gradient boosting.
- `subsample=0.7`: Frazione di campioni utilizzata per addestrare ogni albero, per ridurre l'overfitting.
- `random_state=42`: Seme per la riproducibilità.

Non sono stati riportati test espliciti di *grid search* per questo modello, suggerendo che la configurazione utilizzata rappresenta una scelta di base per il *baseline*. Altri iperparametri, come `max_depth` e `colsample_bytree`, non sono stati specificati nel codice fornito, indicando che sono stati mantenuti ai valori predefiniti della libreria.

#### Note sui Risultati

I risultati delle predizioni sul set di test sono stati raccolti per successive analisi quantitative, come descriverò nella Sezione 5. Le metriche di valutazione specifiche (es. MSE, MAE) e i KPI di business (es. ACoS, ROI) saranno discussi in dettaglio nella Sezione 5, confrontando le performance di `LightGBM` con gli altri modelli testati.

### 3.2.2 XGBoost

Il modello `XGBoost` (*eXtreme Gradient Boosting*) è stato selezionato come *baseline* di regressione per la sua capacità di gestire problemi complessi di predizione attraverso un'implementazione avanzata di gradient boosting. `XGBoost` utilizza alberi decisionali con regolarizzazione L1 e L2 per prevenire l'overfitting, ottimizzando un obiettivo di perdita tramite gradienti di secondo ordine. La sua efficienza e robustezza lo rendono adatto a dataset eterogenei come quello delle campagne PPC descritto nel Capitolo 2, che include feature categoriali, numeriche e ad alta dimensionalità (es. *embedding* SBERT).[12]

#### Implementazione del Modello

Il modello è stato implementato utilizzando la libreria `xgboost` in Python, integrata con `MultiOutputRegressor` di `scikit-learn` per gestire la predizione simultanea di molteplici variabili target: numero di click, spesa, vendite totali a 7 giorni, ordini totali a 7 giorni, tasso di conversione a 7 giorni e impressioni. Le feature utilizzate comprendono variabili categoriali (*lancio*, *performance*, *Day_of_Week*, *Match Type Encoded*, *asin_encoded*), la similarità e 384 *embedding* SBERT, come descritto nel Capitolo 2. È stata applicata una pipeline di preprocessing tramite `ColumnTransformer`, che standardizza (`StandardScaler`) le feature numeriche *Impressions*, *similarity* e *Day_of_Week*, lasciando inalterate le feature SBERT (già normalizzate) e le codifiche categoriali. I dati sono stati suddivisi in un set di addestramento (80%) e un set di test (20%) con `train_test_split` (random seed = 42) per garantire riproducibilità. La pipeline finale integra il preprocessing e il modello `MultiOutputRegressor(XGBRegressor)`, addestrata sui dati di training.

#### Iperparametri Testati

Gli iperparametri configurati per il modello `XGBoost` includono:

- `n_estimators = 150`: numero di alberi di boosting, scelto per bilanciare accuratezza e complessità computazionale.
- `learning_rate = 0.05`: tasso di apprendimento per controllare il contributo di ciascun albero.
- `max_depth = 6`: profondità massima degli alberi per limitare la complessità del modello.
- `subsample = 0.8`: frazione di campioni utilizzata per addestrare ogni albero, per ridurre l'overfitting.
- `colsample_bytree = 0.8`: frazione di feature utilizzate per ogni albero, per aumentare la generalizzazione.

- `random_state = 42`: seed per la riproducibilità.
- `n_jobs = -1`: utilizzo di tutti i core disponibili per accelerare l'addestramento.

Non è stata condotta un'ottimizzazione sistematica degli iperparametri (es. *grid search*) per questo modello, in quanto il suo scopo era fornire una *baseline* robusta per il confronto con modelli più complessi, come descritto nelle sezioni successive.

### 3.2.3 Random Forest

Il modello `Random Forest` è stato selezionato come *baseline* di regressione per la sua semplicità, robustezza al rumore nei dati e capacità di gestire feature eterogenee, come quelle descritte nel Capitolo 2. `Random Forest` è un metodo di ensemble che combina molteplici alberi decisionali, ciascuno addestrato su un sottoinsieme casuale di dati e feature, utilizzando il *bagging* (Bootstrap Aggregating) per ridurre la varianza e migliorare la generalizzazione. La sua natura non parametrica lo rende adatto a modellare relazioni complesse tra le feature (es. *embedding* SBERT, similarità, variabili categoriali) e i target multipli delle campagne PPC (click, spesa, vendite, ecc.).

#### Implementazione del Modello

Il modello è stato implementato utilizzando la libreria `sklearn.ensemble.RandomForestRegressor` in Python, integrata con `MultiOutputRegressor` di `scikit-learn` per gestire la predizione simultanea di molteplici variabili target: numero di click, spesa, vendite totali a 7 giorni, ordini totali a 7 giorni, tasso di conversione a 7 giorni e impressioni. Le feature utilizzate includono variabili categoriali (*lancio*, *performance*, *Day_of_Week*, *Match Type Encoded*), la similarità e 384 *embedding* SBERT, come descritto nel Capitolo 2. È stata applicata una pipeline di preprocessing tramite `ColumnTransformer`, che standardizza (`StandardScaler`) le feature numeriche *Impressions*, *similarity* e *Day_of_Week*, lasciando inalterate le feature SBERT (già normalizzate) e le codifiche categoriali. I dati sono stati suddivisi in un set di addestramento (80%) e un set di test (20%) con `train_test_split` (random seed = 42) per garantire riproducibilità. La pipeline finale integra il preprocessing e il modello `MultiOutputRegressor(RandomForestRegressor)`, addestrata sui dati di training.

#### Iperparametri Testati

Gli iperparametri configurati per il modello `Random Forest` includono:

- `n_estimators = 100`: numero di alberi nell'ensemble, scelto per bilanciare accuratezza e tempi di calcolo.
- `max_depth = 15`: profondità massima degli alberi per limitare la complessità e prevenire l'overfitting.
- `random_state = 42`: seed per la riproducibilità.
- `n_jobs = -1`: utilizzo di tutti i core disponibili per accelerare l'addestramento.

Nel codice fornito, i parametri `learning_rate` e `colsample_bytree` sono indicati come non definiti (`n/d`), in quanto non applicabili a `Random Forest`. Analogamente, il

parametro `subsample` è stato interpretato come `1 - test_size = 0.8`, riflettendo la proporzione dei dati di addestramento. Non è stata condotta un'ottimizzazione sistematica degli iperparametri (es. *grid search*) per questo modello, in quanto il suo scopo era fornire una *baseline* robusta per il confronto con modelli più complessi, come descritto nelle sezioni successive.

#### Fonti

La descrizione del funzionamento di `Random Forest` si basa sulla documentazione ufficiale di `scikit-learn` (`https://scikit-learn.org/`) e su [8], che introduce l'algoritmo e le sue proprietà di ensemble. L'implementazione con `MultiOutputRegressor` e `ColumnTransformer` si rifà alla documentazione di `scikit-learn`.

## 3.3 Reti Multi-Layer Perceptron

### 3.3.1 Reti MLP e Ottimizzazione

Le reti *Multi-Layer Perceptron* (`MLP`) sono state scelte per la loro capacità di modellare relazioni non lineari complesse tra le feature e i target multipli delle campagne PPC, come descritto nel Capitolo 2. Le `MLP` sono reti neurali *feed-forward* composte da strati di neuroni interconnessi, in grado di apprendere rappresentazioni non lineari tramite funzioni di attivazione e ottimizzazione degli iperparametri. Tre configurazioni di `MLP` sono state sperimentate, variando architetture, meccanismi di regolarizzazione e tecniche di addestramento per ottimizzare le prestazioni predittive. Le sezioni seguenti descrivono in dettaglio le fasi di pre-elaborazione comuni, l'implementazione di ciascun modello, il ruolo di ogni strato e gli iperparametri testati.

#### Pre-elaborazione dei Dati

Le fasi di pre-elaborazione sono state progettate per preparare i dati per l'addestramento delle `MLP`, garantendo stabilità numerica e rappresentazione efficace delle feature. Le feature utilizzate includono variabili categoriali (`lancio`, `performance`, `Day_of_Week`, `Match Type Encoded`), la similarità e 384 *embedding* SBERT, come descritto nel Capitolo 2. I target considerati sono: impressioni, numero di click, spesa, vendite totali a 7 giorni, ordini totali a 7 giorni, tasso di conversione a 7 giorni e impressioni. Di seguito, le fasi di pre-elaborazione comuni a tutte le `MLP`:

- **Codifica delle feature categoriali**: La feature `Day_of_Week` è stata codificata utilizzando `LabelEncoder` di `scikit-learn` per trasformare i valori categoriali in numerici ordinali, garantendo compatibilità con le operazioni matriciali delle `MLP`. Questo passaggio è essenziale per rappresentare le informazioni temporali in modo numerico [40].
- **Riduzione dimensionale degli embedding SBERT** (solo per `MLP1`): Gli *embedding* SBERT a 384 dimensioni sono stati ridotti a 50 dimensioni utilizzando `UMAP` (`umap-learn`), un algoritmo di riduzione dimensionale non lineare che preserva la struttura locale dei dati. UMAP è stato configurato con `n_components=50`, `random_state=42` e `n_jobs=1` per riproducibilità e controllo computazionale. Successivamente, è stato applicato il clustering HDBSCAN (`hdbscan`) con `min_cluster_size=50`

e `min_samples=10` per identificare pattern nei dati combinando gli embedding ridotti con feature scalate (`Spend`, `Clicks`, `7 Day Total Sales`). La feature `Cluster` risultante è stata aggiunta al dataset, arricchendo il contesto per l'`MLP1` [34, 10].

- **Scalatura delle feature non-SBERT**: Le feature `Impressions`, `lancio`, `performance`, `Day_of_Week`, `Match Type Encoded` e `similarity` sono state standardizzate con `StandardScaler` di `scikit-learn` per garantire media zero e varianza unitaria, migliorando la stabilità dell'addestramento. Gli embedding SBERT, già normalizzati, sono stati lasciati inalterati. Il modello `StandardScaler` è stato salvato in un file `stdScaler.pkl` per utilizzi futuri [40].

- **Trasformazione logaritmica dei target**: I target sono stati trasformati con `np.log1p` per stabilizzare la varianza e gestire distribuzioni asimmetriche, come tipico per dati di campagne PPC (es. spesa e vendite). Le predizioni sono state ritrasformate con `np.expm1` per ottenere valori nella scala originale [24].

- **Conversione in tensori PyTorch**: Le feature e i target sono stati convertiti in tensori PyTorch di tipo `float32` e trasferiti su dispositivo (`device`, es. GPU) per sfruttare l'accelerazione hardware [39].

- **Split dei dati**: I dati sono stati suddivisi in set di addestramento, validazione e test utilizzando `train_test_split` di `scikit-learn`. Per `MLP1`, è stato utilizzato uno split 70%-15%-15%, mentre per `MLP2` e `MLP3` uno split 70%-15%-15% (train-val-test). Il parametro `random_state=42` garantisce riproducibilità [40].

Il codice di pre-elaborazione per `MLP1` è riportato di seguito:

```
from sklearn.preprocessing import Label>>>Encoder, StandardScaler
from umap import UMAP
from hdbscan import HDBSCAN
import numpy as np
import torch
import pickle

le = LabelEncoder()
df['Day_of_Week'] = le.fit_transform(df['Day_of_Week'])
features = ['lancio', 'performance', 'Day_of_Week', 'Match Type
    Encoded', 'similarity'] + [f'sbert_{i}' for i in range(384)]
targets = ['Clicks', 'Spend', '7 Day Total Sales', '7 Day Total Orders
    (#)', '7 Day Conversion Rate', 'Impressions']
sbert_features = [f'sbert_{i}' for i in range(384)]

# Riduzione dimensionale e clustering (solo MLP1)
umap = UMAP(n_components=50, random_state=42, n_jobs=1)
sbert_embeddings = df[sbert_features].values
sbert_reduced = umap.fit_transform(sbert_embeddings)
scaler_performance = StandardScaler()
performance_features = ['Spend', 'Clicks', '7 Day Total Sales']
performance_data = df[performance_features].values
performance_scaled = scaler_performance.fit_transform(performance_data)
clustering_data = np.hstack([sbert_reduced, performance_scaled])
hdbscan_model = HDBSCAN(min_cluster_size=50, min_samples=10)
cluster_labels = hdbscan_model.fit_predict(clustering_data)
df['Cluster'] = cluster_labels
features += ['Cluster']
```

```
df.to_csv("dataset_tmp_cluster.csv")

# Scalatura feature non-SBERT
scaler = StandardScaler()
non_sbert_features = ['lancio', 'performance', 'Day_of_Week', 'Match
    Type Encoded', 'similarity']
X = df[features].values
X[:, :len(non_sbert_features)] = scaler.fit_transform(X[:,
    :len(non_sbert_features)])
y = np.log1p(df[targets].values)

# Salva scaler
with open("modelli/stdScaler.pkl", "wb") as f:
    pickle.dump(scaler, f)

# Converti in tensori
X = torch.tensor(X, dtype=torch.float32).to(device)
y = torch.tensor(y, dtype=torch.float32).to(device)

# Split Train-Val-Test
X_train, X_temp, y_train, y_temp = train_test_split(X, y,
    test_size=0.1, random_state=42, shuffle=True)
```

Per `MLP2` e `MLP3`, il codice di pre-elaborazione è simile, senza il clustering HDBSCAN:

```
le = LabelEncoder()
df['Day_of_Week'] = le.fit_transform(df['Day_of_Week'])
features = ['lancio', 'performance', 'Day_of_Week', 'Match Type
    Encoded', 'similarity'] + [f'sbert_{i}' for i in range(384)]
targets = ['Clicks', 'Spend', '7 Day Total Sales', '7 Day Total Orders
    (#)', '7 Day Conversion Rate', 'Impressions']
X = df[features].values
y = df[targets].values

# Scalatura feature non-SBERT
scaler = StandardScaler()
non_sbert_features = ['lancio', 'performance', 'Day_of_Week', 'Match
    Type Encoded', 'similarity']
X[:, :len(non_sbert_features)] = scaler.fit_transform(X[:,
    :len(non_sbert_features)])
y = np.log1p(y)

# Converti in tensori
X = torch.tensor(X, dtype=torch.float32).to(device)
y = torch.tensor(y, dtype=torch.float32).to(device)

# Split Train-Val-Test
X_train, X_temp, y_train, y_temp = train_test_split(X, y,
    test_size=0.3, random_state=42, shuffle=True)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
    test_size=0.5, random_state=42)
```

### Primo Modello MLP

**Descrizione e Funzionamento** Il primo modello MLP è una rete feed-forward con tre strati nascosti, progettata per catturare relazioni non lineari tra feature e target multipli. La Huber loss (`nn.HuberLoss`) è stata scelta per la sua robustezza agli outlier rispetto alla MSE [25]. La struttura è definita come segue:

- **Strato di input** (`input_dim=390 o 391`): Accetta le feature, inclusa `Cluster` derivata dal clustering. Questo strato mappa l'input ad alta dimensionalità (es. 384 *embedding* SBERT) al primo strato nascosto.
- **Primo strato nascosto** (`512 neuroni`): Riduce la dimensionalità dell'input e apprende rappresentazioni intermedie complesse tramite `ReLU`, che introduce non linearità per catturare pattern non lineari [36]. La dimensione elevata (512) consente di esplorare un ampio spazio di rappresentazioni.
- **Secondo strato nascosto** (`256 neuroni`): Raffina ulteriormente le rappresentazioni, riducendo la complessità e focalizzandosi su feature rilevanti. `ReLU` mantiene la non linearità.
- **Terzo strato nascosto** (`128 neuroni`): Compatta le informazioni per preparare l'output, bilanciando capacità espressiva e rischio di overfitting.
- **Strato di output** (`output_dim=6`): Produce predizioni per i sei target, senza attivazione finale per consentire valori continui.

**Implementazione** Il modello è stato implementato in PyTorch con la classe `MLP`, che costruisce una rete sequenziale di strati lineari e attivazioni ReLU. L'addestramento utilizza l'ottimizzatore Adam (`lr=0.0001`) e la HuberLoss. I dati sono stati suddivisi in training (70%) e test (30%, usato come validation), con batch di dimensione 256 e 550 epoche. Una barra di avanzamento (`tqdm`) monitora la perdita di validazione e stima il tempo rimanente. Il codice è il seguente:

```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim):
        super(MLP, self).__init__()
        layers = []
        prev_dim = input_dim
        for dim in hidden_dims:
            layers.append(nn.Linear(prev_dim, dim))
            layers.append(nn.ReLU())
            prev_dim = dim
        layers.append(nn.Linear(prev_dim, output_dim))
        self.network = nn.Sequential(*layers)

    def forward(self, x):
        return self.network(x)

input_dim = X_train.shape[1]
hidden_dims = [512, 256, 128]
output_dim = y_train.shape[1]
model = MLP(input_dim, hidden_dims, output_dim).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
criterion = nn.HuberLoss()
epochs = 550
batch_size = 256
train_model(model, X_train, y_train, X_temp, y_temp, epochs,
   batch_size)
```

**Iperparametri Testati** Gli iperparametri includono:

- `hidden_dims = [512, 256, 128]`: Configurazione degli strati nascosti per bilanciare capacità e complessità.
- `batch_size = 256`: Dimensione del batch per stabilizzare l'addestramento.
- `epochs = 550`: Numero di epoche per consentire convergenza.
- `optimizer = Adam, lr = 0.0001`: Ottimizzatore e tasso di apprendimento bassi per una convergenza graduale.
- `criterion = HuberLoss`: Perdita robusta agli outlier.

Non sono state applicate regolarizzazione, normalizzazione batch, dropout, scheduler o early stopping.

**Fonti** HuberLoss è descritta in [25]. L'implementazione in PyTorch si basa sulla documentazione ufficiale (`https://pytorch.org/`). La funzione ReLU è tratta da [36]. Il clustering UMAP e HDBSCAN è supportato da [34, 10].

### Secondo Modello MLP

**Descrizione e Funzionamento** Il secondo modello MLP introduce regolarizzazioni (dropout, batch normalization) e un scheduler per migliorare la generalizzazione e la convergenza. Utilizza una perdita MSE ponderata per bilanciare i target con diverse scale. La struttura è:

- **Strato di input (`input_dim=390`)**: Accetta le feature senza `Cluster`, con 384 *embedding* SBERT.
- **Primo strato nascosto (`256 neuroni`)**: Mappa l'input a una rappresentazione intermedia, con `BatchNorm1d` per stabilizzare la distribuzione delle attivazioni [27], `ReLU` [36] e `Dropout(0.2)` per prevenire l'overfitting [46].
- **Secondo strato nascosto (`128 neuroni`)**: Raffina le rappresentazioni, con `BatchNorm1d`, `ReLU` e `Dropout(0.2)` per mantenere la generalizzazione.
- **Terzo strato nascosto (`64 neuroni`)**: Compatta le informazioni, con le stesse reg introduzioni per robustezza.
- **Strato di output (`output_dim=6`)**: Predice i target senza attivazione finale.

**Implementazione** Il modello è implementato in PyTorch con `BatchNorm1d` e `Dropout` in ogni strato nascosto. La perdita MSE (`nn.MSELoss`) è ponderata inversamente alla media dei target per bilanciare le scale. L'addestramento utilizza Adam (`lr=0.001`), un scheduler `ReduceLROnPlateau` (`factor=0.5`, `patience=10`) e early stopping (`patience=20`). I dati sono suddivisi in training (70%), validation (15%) e test (15%), con batch di dimensione 512 e 100 epoche. Il codice è:

```
weights = torch.tensor(1.0 / (np.mean(y, axis=0) + 1e-6),
    dtype=torch.float32).to(device)
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim):
        super(MLP, self).__init__()
        layers = []
        prev_dim = input_dim
        for dim in hidden_dims:
            layers.append(nn.Linear(prev_dim, dim))
            layers.append(nn.BatchNorm1d(dim))
            layers.append(nn.ReLU())
            leggi.append(nn.Dropout(0.2))
            prev_dim = dim
        layers.append(nn.Linear(prev_dim, output_dim))
        self.network = nn.Sequential(*layers)

    def forward(self, x):
        return self.network(x)

input_dim = X_train.shape[1]
hidden_dims = [256, 128, 64]
output_dim = y_train.shape[1]
model = MLP(input_dim, hidden_dims, output_dim).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.MSELoss(reduction='none')
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
    mode='min', factor=0.5, patience=10)
epochs = 100
batch_size = 512
model = train_model(model, X_train, y_train, X_val, y_val, epochs,
    batch_size, weights)
```

### Iperparametri Testati

- `hidden_dims = [256, 128, 64]`: Strati più compatti rispetto al primo modello.
- `batch_size = 512`: Batch più grande per stabilità.
- `epochs = 100`: Numero ridotto grazie a early stopping.
- `optimizer = Adam, lr = 0.001`: Tasso di apprendimento più alto.
- `dropout = 0.2`: Applicato a ogni strato nascosto.
- `batch_normalization = True`: `BatchNorm1d` per ogni strato.
- `scheduler = ReduceLROnPlateau`: Riduce il learning rate se la perdita di validazione non migliora.
- `early_stopping = 20`: Ferma l'addestramento dopo 20 epoche senza miglioramenti.
- `criterion = MSELoss (weighted)`: Perdita ponderata per bilanciare i target.

**Fonti** La batch normalization è descritta in [27], il dropout in [46], lo scheduler in `ReduceLROnPlateau` e l'implementazione MLP si basa su `https://pytorch.org/`.

### Terzo Modello MLP

**Descrizione e Funzionamento** Il terzo modello MLP introduce `LayerNorm` al posto di `BatchNorm1d`, un dropout ridotto e `weight_decay` per una regolarizzazione più leggera, insieme al gradient clipping per migliorare la stabilità dell'addestramento. HuberLoss è utilizzata per robustezza. La struttura è:

- **Strato di input** (`input_dim=390`): Accetta le feature senza `Cluster`.
- **Primo strato nascosto** (`256 neuroni`): Mappa l'input con `LayerNorm` per normalizzare le attivazioni all'interno di ciascun esempio [5], `ReLU` [36] e `Dropout(0.1)` [46].
- **Secondo strato nascosto** (`128 neuroni`): Raffina le rappresentazioni con `LayerNorm`, `ReLU` e `Dropout(0.1)`.
- **Terzo strato nascosto** (`64 neuroni`): Compatta le informazioni, con le stesse tecniche di regolarizzazione.
- **Strato di output** (`output_dim=6`): Predice i target.

**Implementazione** Implementato in PyTorch con `LayerNorm`, `Dropout(0.1)` e con `weight_decay = 1e-5` nell'ottimizzatore Adam (`lr=0.005`). Il gradient clipping (`max_norm = 1.0`) previene esplosioni del gradiente. HuberLoss(`reduction='mean'`) è utilizzata. I dati sono suddivisi in training (70%), validation (15%) e test (15%), con batch di dimensione 512 e 1000 epoche, con early stopping (`patience=30`). Il codice è:

```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim):
        super(MLP, self).__init__()
        layers = []
        prev_dim = input_dim
        for dim in hidden_dims:
            layers.append(nn.Linear(prev_dim, dim))
            layers.append(nn.LayerNorm(dim))
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(0.1))
            prev_dim = dim
        layers.append(nn.Linear(prev_dim, output_dim))
        self.network = nn.Sequential(*layers)

    def forward(self, x):
        return self.network(x)

input_dim = X_train.shape[1]
hidden_dims = [256, 128, 64]
output_dim = y_train.shape[1]
model = MLP(input_dim, hidden_dims, output_dim).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.005,
   weight_decay=1e-5)
criterion = nn.HuberLoss(reduction='mean')
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
   mode='min', factor=0.7, patience=15)
epochs = 1000
batch_size = 512
```

```
model = train_model(model, X_train, y_train, X_val, y_val, epochs,
    batch_size, patience=30)
```

**Iperparametri Testati**

- `hidden_dims = [256, 128, 64], [512, 256, 128], [1024, 512, 256]`: Strati compatti.
- `batch_size = da 256, 512 fino a 1024]`: Batch grande per stabilità.
- `epochs = da 100 a 700`: Massimo,con incrementi di 50, con early stopping.
- Regolarizzazione L2.
  - `optimizer = Adam,RMSprop,AdamW`
  - `lr = da 0.005 a 0.0000001`
  - `weight_decay = 1e-5`
- `layer_normalization = True`: `LayerNorm` per ogni strato.
- `scheduler = ReduceLROnPlateau`: Fattore 0.7, `patience=15`.
- `early_stopping = 30`: Ferma dopo 30 epoche senza miglioramenti.
- `criterion = HuberLoss`: Robusta agli outlier.

**Fonti** La `LayerNorm` è descritta in [5], il dropout in [46], HuberLoss in [25], e l'implementazione in PyTorch su `https://pytorch.org/`.

## 3.4 Contextual Bandit

### 3.4.1 Introduzione al Multi-Armed Bandit

Il problema del *Multi-Armed Bandit* (MAB) è un framework classico di reinforcement learning, utilizzato per modellare decisioni in ambienti con incertezza, bilanciando l'esplorazione di opzioni sconosciute e lo sfruttamento delle opzioni note per massimizzare una ricompensa cumulativa [44]. Il termine deriva dall'analogia con un giocatore d'azzardo che deve scegliere tra più slot machine ("banditi a un braccio"), ciascuna con una distribuzione di ricompense sconosciuta, decidendo quale leva tirare per ottenere il massimo guadagno. Formalmente, un MAB è modellato come un insieme di $K$ "bracci" (azioni), ciascuno associato a una distribuzione di ricompense sconosciuta con valore atteso $\mu_k$. L'agente sceglie un braccio $a_t$ al tempo $t$, osserva una ricompensa $r_t$ e aggiorna la propria stima della distribuzione per massimizzare la ricompensa cumulativa su un orizzonte temporale $T$. La sfida principale è il compromesso tra *esplorazione* (provare bracci per raccogliere informazioni) e *sfruttamento* (scegliere il braccio con la migliore ricompensa stimata) [48].

### Funzionamento

Il funzionamento di un MAB si basa su algoritmi che bilanciano esplorazione e sfruttamento. Tra i più comuni troviamo:

- **Epsilon-Greedy**: Sceglie il braccio con la ricompensa media ottimale (basandosi sulle attuali conoscenze) con probabilità $1 - \epsilon$ e un braccio casuale con probabilità $\epsilon$, garantendo esplorazione. In breve l'algoritmo $\epsilon$-greedy segue il seguente processo decisionale [48]:

$$A_t = \begin{cases} \max Q_t(a) & \text{(con probabilità } 1 - \epsilon) \\ \text{Random}(a) & \text{(con probabilità } \epsilon) \end{cases} \tag{3.1}$$

**Figura 3.1:** Dove $Q_t(a)$ è la stima della ricompensa attesa al tempo $t$ per l'azione $a$ mentre $A_t$ è l'azione al tempo $t$

- **Upper Confidence Bound (UCB)**: Seleziona il braccio con il massimo valore di una stima ottimistica, combinando la ricompensa media con un termine di incertezza proporzionale al numero di volte che il braccio è stato scelto [4].
- **Thompson Sampling**: Usa un approccio bayesiano, campionando le ricompense attese da una distribuzione a posteriori e scegliendo il braccio con il valore campionato più alto [49].

Il *regret*, definito come la differenza tra la ricompensa attesa di una strategia ottimale e quella ottenuta, è una metrica chiave per valutare le prestazioni degli algoritmi MAB [32]. L'obiettivo è minimizzare il *regret* su $T$ iterazioni.

### Usi Comuni

I MAB trovano applicazione in scenari in cui è necessario ottimizzare decisioni in tempo reale con risorse limitate, come:

- **Ottimizzazione di contenuti online**: Selezione dinamica di articoli o annunci per massimizzare clic o engagement [38].
- **Test clinici**: Allocazione di pazienti a trattamenti sperimentali per ottimizzare i risultati minimizzando i fallimenti [18].
- **Routing adattivo**: Ottimizzazione dei percorsi in reti per ridurre i tempi di latenza [53].
- **Gestione di progetti di ricerca**: Allocazione di risorse in organizzazioni come fondazioni scientifiche o aziende farmaceutiche [53].

**Applicazioni Reali in Aziende**

- **Google Analytics**: Google ha utilizzato MAB per ottimizzare la visualizzazione dei risultati di ricerca, bilanciando l'esplorazione di nuove configurazioni con lo sfruttamento di quelle più efficaci [29].
- **The Washington Post**: Utilizza MAB per testare titoli, miniature di immagini e articoli suggeriti, massimizzando i clic in finestre temporali brevi, sfruttando la natura time-sensitive delle notizie [23].
- **Stitch Fix**: Implementa MAB nella sua piattaforma di sperimentazione per ottimizzare le raccomandazioni di prodotti, utilizzando metriche come il *lifetime value* (LTV) per valutare le scelte [17].
- **Udemy**: Applica MAB per il ranking delle unità di raccomandazione (es. caroselli di corsi), affrontando il problema di ordinamento come un MAB per migliorare l'engagement degli utenti [50].

### 3.4.2 Contextual Bandit: Definizione e Differenze

I *Contextual Bandit* (CB) sono un'estensione dei MAB che incorporano informazioni contestuali per personalizzare le decisioni. In un CB, l'agente riceve un vettore di contesto $v_t \in \mathbb{R}^d$ al tempo $t$, che descrive caratteristiche dell'ambiente o dell'utente (es. dati demografici, preferenze, comportamento passato). L'agente sceglie un braccio $a_t$ in base al contesto, osserva una ricompensa $r_t$ e aggiorna il modello per prevedere la ricompensa attesa per ciascun braccio dato il contesto [33]. La differenza principale rispetto ai MAB è che i CB modellano la dipendenza tra ricompense e contesto, consentendo decisioni personalizzate piuttosto che una singola scelta ottimale per tutti gli utenti [45].

**Funzionamento**

I CB utilizzano algoritmi che combinano il contesto con le stime delle ricompense. Tra i più noti:

- **LinUCB**: Assume una relazione lineare tra contesto e ricompensa attesa, utilizzando un intervallo di confidenza per bilanciare esplorazione e sfruttamento [33].
- **LinRel**: Simile a LinUCB, ma usa la decomposizione ai valori singolari per stimare l'incertezza invece della regressione ridge [53].
- **UCBogram**: Stima funzioni di ricompensa non lineari tramite un estimatore a tratti costanti (regressogramma) [53].
- **Oracle-based**: Riduce il problema a una serie di problemi di apprendimento supervisionato, senza assumere una forma specifica della funzione di ricompensa [53].

Il *regret* in un CB è definito rispetto alla strategia ottimale che sceglie il miglior braccio per ciascun contesto, rendendo il problema più complesso ma più flessibile rispetto ai MAB [45].

**Differenze rispetto al Multi-Armed Bandit**

- **Incorporazione del contesto**: I MAB cercano un unico braccio ottimale per tutti gli utenti, mentre i CB selezionano il braccio migliore per ciascun contesto, consentendo personalizzazione [38].
- **Complessità computazionale**: I CB richiedono modelli più complessi per gestire vettori di contesto ad alta dimensionalità, aumentando il costo computazionale rispetto ai MAB [37].
- **Personalizzazione**: I CB sfruttano informazioni contestuali (es. caratteristiche dell'utente o dell'azione) per decisioni 1:1, mentre i MAB si basano solo sulle ricompense passate [38].
- **Applicazioni**: I MAB sono adatti a problemi con ricompense stazionarie e senza contesto, mentre i CB eccellono in scenari dinamici e personalizzati [45].

**Applicazioni Reali di Contextual Bandit**

- **Instacart**: Utilizza CB per personalizzare le raccomandazioni di prodotti in base al contesto dell'utente (es. posizione, storico acquisti), affrontando la sfida di uno spazio di feature ampio [47].
- **Stitch Fix**: Integra CB nella sua piattaforma per raccomandazioni personalizzate di abbigliamento, utilizzando il contesto delle preferenze degli utenti per migliorare il *lifetime value* [17].
- **Azienda di ridesharing (non specificata)**: Ha implementato CB per personalizzare promozioni per conducenti e passeggeri, richiedendo un team di 50 ingegneri senior per gestire la complessità [37].
- **Optimizely**: Usa CB per fornire esperienze personalizzate su siti web, adattando contenuti in tempo reale in base a caratteristiche come dispositivo, posizione o comportamento dell'utente [38].

### 3.4.3 Implementazione Contextual Bandit

Il modello *Contextual Bandit* (CB) è stato adottato per affrontare il problema di allocazione dinamica dei budget pubblicitari nelle campagne PPC di Amazon, bilanciando esplorazione e sfruttamento per ottimizzare la ricompensa attesa, definita come una combinazione di vendite e costi. Questo modello si basa sull'integrazione con una rete *Multi-Layer Perceptron* (MLP) precedentemente addestrata, descritta nella Sezione 3.3.1, per inizializzare le stime delle ricompense. Questa sezione descrive il funzionamento generale del *Contextual Bandit*, il processo di implementazione, i dettagli specifici del modello utilizzato e la pipeline completa, con particolare attenzione agli aspetti del bandit, evitando di approfondire l'MLP, che sarà trattato nel capitolo successivo.

**Funzionamento del Contextual Bandit**

Un *Contextual Bandit* è un'estensione del problema *Multi-Armed Bandit* (MAB), che incorpora informazioni contestuali per personalizzare le decisioni. In questo contesto,

ogni "braccio" rappresenta un'azione possibile (es. una strategia di bidding), e il vettore di contesto descrive caratteristiche specifiche della campagna PPC, come `lancio`, `performance`, `Day_of_Week`, `Match Type Encoded`, `similarity`, `Cluster` e 384 *embedding* SBERT, come descritto nel Capitolo 2.2. L'obiettivo è selezionare l'azione che massimizza la ricompensa attesa, definita come (3.2), dove $k$ è un fattore di penalizzazione per i costi (nel codice, $k = 0.5$). Il modello utilizza un approccio basato su Thompson Sampling, campionando le ricompense attese da distribuzioni gaussiane per ogni azione e aggiornando le stime in base alle ricompense osservate [49].

$$R = sales - k * spent \tag{3.2}$$

**Figura 3.2:** Formula per la stima del reward atteso ($R$), dove *sales* rappresenta le vendite generate e *spent* la spesa pubblicitaria sostenuta per una determinata keyword in un dato giorno di campagna. Il parametro $k$ regola l'entità della penalizzazione associata ai costi.

Il funzionamento si articola in tre fasi principali:

- **Inizializzazione**: Le stime iniziali delle ricompense per ciascun braccio sono calcolate utilizzando le predizioni dell'MLP su un set di dati, selezionando i contesti con ricompense superiori al quantile 0.8 per inizializzare i parametri delle distribuzioni gaussiane (`mu` e `sigma`).
- **Selezione delle azioni**: Per ogni contesto, il modello campiona ricompense attese da una distribuzione normale per ogni azione e sceglie l'azione con la ricompensa campionata più alta. Questo approccio bayesiano bilancia esplorazione e sfruttamento [11].
- **Aggiornamento**: Le distribuzioni gaussiane (`mu` e `sigma`) di ciascun braccio vengono aggiornate con una media ponderata basata sui contesti selezionati, garantendo che il modello si adatti dinamicamente ai nuovi dati osservati [32].

Il *regret*, definito come la differenza tra la ricompensa massima possibile e quella ottenuta, è calcolato per valutare le prestazioni del modello. La pipeline include anche una funzione per identificare il contesto ottimale, che massimizza la ricompensa attesa, utile per suggerire strategie di bidding ottimali.

### Implementazione del Modello

Il modello *Contextual Bandit* è stato implementato in PyTorch come una classe `ContextualBandit`, che definisce un insieme di modelli lineari (uno per ogni azione) per stimare le ricompense attese in base al contesto. La pipeline include l'inizializzazione con l'MLP, l'addestramento del bandit, il calcolo del *regret* e la ricerca del contesto ottimale. Di seguito, i dettagli specifici del modello e della pipeline.

**Struttura del Modello** La classe `ContextualBandit` è definita con i seguenti componenti:

- **Input**: Un vettore di contesto di dimensione `context_dim` (es. 391, incluse le feature e la `Cluster` derivata dal clustering).

- **Modelli lineari**: Una lista di `num_actions` modelli lineari (`nn.Linear`), ciascuno mappante il contesto a una ricompensa scalare per un'azione specifica.
- **Distribuzioni gaussiane**: Per ogni azione, i parametri `mu` (media) e `sigma` (deviazione standard) modellano la distribuzione delle ricompense attese, utilizzate per il campionamento in stile Thompson Sampling.
- **Metodo `forward`**: Calcola le ricompense attese per tutte le azioni dato un contesto.
- **Metodo `sample_rewards`**: Campiona ricompense da distribuzioni gaussiane per ogni azione, utilizzando `torch.normal` per implementare Thompson Sampling.

Il codice della classe `ContextualBandit` è:

```
class ContextualBandit(nn.Module):
    def __init__(self, context_dim, num_actions):
        super(ContextualBandit, self).__init__()
        self.num_actions = num_actions
        self.context_dim = context_dim
        self.action_models = nn.ModuleList([nn.Linear(context_dim, 1)
            for _ in range(num_actions)])
        self.mu = torch.zeros(num_actions, context_dim)
        self.sigma = torch.ones(num_actions, context_dim)

    def forward(self, context):
        rewards = torch.stack([model(context) for model in
            self.action_models], dim=1).squeeze(-1)
        return rewards

    def sample_rewards(self, context):
        sampled_rewards = torch.zeros(context.size(0),
            self.num_actions).to(context.device)
        for a in range(self.num_actions):
            sampled_mu =
                torch.normal(mean=self.mu[a].to(context.device),
                std=self.sigma[a].to(context.device))
            sampled_rewards[:, a] = torch.matmul(context, sampled_mu)
        return sampled_rewards
```

**Inizializzazione con MLP** L'inizializzazione del bandit utilizza le predizioni dell'MLP per calcolare una ricompensa iniziale basata su `sales - k * spend`. I contesti con ricompense superiori al quantile 0.8 sono utilizzati per stimare `mu` (media) e `sigma` (deviazione standard) per ogni azione. Questo approccio di *warm-start* accelera la convergenza rispetto a un'inizializzazione casuale [22]. Il codice è:

```
def initialize_bandit_with_mlp(bandit, mlp_model, X, k, device):
    mlp_model.eval()
    X = X.to(device)
    with torch.no_grad():
        y_pred = mlp_model(X)
        rewards = compute_reward(y_pred, k)
        for a in range(bandit.num_actions):
            bandit.mu[a] = torch.mean(X[rewards >
                rewards.quantile(0.8)].cpu(), dim=0)
            bandit.sigma[a] = torch.std(X[rewards >
                rewards.quantile(0.8)].cpu(), dim=0) + 1e-6
```

**Funzione di Ricompensa** La ricompensa è calcolata come Vendite $- k \cdot$ Spesa, con $k = 0.5$, per bilanciare l'ottimizzazione dei ricavi con la minimizzazione dei costi. La funzione `compute_reward` ritrasforma i target predetti dall'MLP (`np.expm1`) per ottenere valori nella scala originale:

```
def compute_reward(y, k=0.5):
    y = np.expm1(y.cpu().numpy())
    sales = y[:, 2]
    spend = y[:, 1]
    reward = sales - k * spend
    return torch.tensor(reward, dtype=torch.float32)
```

**Addestramento** L'addestramento del bandit utilizza l'ottimizzatore Adam (`lr` specificato come parametro) e la perdita MSE per minimizzare la differenza tra le ricompense previste e quelle osservate. Per ogni batch, il modello campiona azioni tramite `sample_rewards`, calcola la perdita e aggiorna i parametri dei modelli lineari e le distribuzioni gaussiane (`mu`, `sigma`) con una media ponderata (90% valore precedente, 10% nuovo). Il codice è:

```
def train_bandit(bandit, X, y, epochs, batch_size, lr, k, device):
    bandit = bandit.to(device)
    optimizer = torch.optim.Adam(bandit.parameters(), lr=lr)
    criterion = nn.MSELoss()
    bandit.train()
    rewards_history = []
    actions_history = []

    X = X.to(device)
    y = y.to(device)

    for epoch in tqdm(range(epochs), desc=f"Addestramento Bandit
        (lr={lr}, batch={batch_size})"):
        indices = torch.randperm(X.shape[0])
        for start in range(0, X.shape[0], batch_size):
            batch_indices = indices[start:start + batch_size]
            X_batch = X[batch_indices]
            y_batch = y[batch_indices]

            optimizer.zero_grad()
            sampled_rewards = bandit.sample_rewards(X_batch)
            actions = torch.argmax(sampled_rewards, dim=1)
            rewards = compute_reward(y_batch, k).to(device)
            predicted_rewards = bandit(X_batch)
            selected_rewards =
               predicted_rewards[torch.arange(X_batch.size(0)),
               actions]
            loss = criterion(selected_rewards, rewards)
            loss.backward()
            optimizer.step()

            for a in range(bandit.num_actions):
                mask = actions == a
                if mask.sum() > 0:
                    bandit.mu[a] = 0.9 * bandit.mu[a] + 0.1 *
                       torch.mean(X_batch[mask].cpu(), dim=0)
                    bandit.sigma[a] = 0.9 * bandit.sigma[a] + 0.1 *
                       (torch.std(X_batch[mask].cpu(), dim=0) + 1e-6)
```

```
            rewards_history.append(rewards.mean().item())
            actions_history.append(actions.cpu().numpy())

    return rewards_history, actions_history
```

**Calcolo del Regret** La funzione `calculate_regret(bandit, X, y, k, device)` implementa la misura del *regret* cumulativo associato a una politica di bandit contestuale valutata su un dataset di contesti $\mathbf{X}$ e osservazioni $\mathbf{y}$. Di seguito si fornisce una descrizione dettagliata, passo per passo, del comportamento e delle operazioni effettuate dal codice.

- **Input attesi:** `bandit` è un modello (tipicamente una sottoclasse di `torch.nn.Module`) che, dato un batch di contesti $X \in \mathbb{R}^{N \times d}$, restituisce una matrice di ricompense predette con shape $(N, A)$, dove $A$ è il numero di azioni. `X` e `y` sono tensori PyTorch contenenti rispettivamente i contesti e i target osservati; `k` è un parametro scalare usato nella funzione di calcolo della ricompensa; `device` indica la destinazione (CPU/GPU).
- **Variabile esterna:** il codice fa riferimento a una variabile `version` che decide quale procedura usare; la variabile viene settata in ambiente globale e dirige l'esecuzione del codice allenando il bandit con un calcolo del regret differente a seconda della versione scelta. Questa variabile cambia drasticamente il comportamento del bandit, come verrà ampiamente spiegato nel Capitolo 4, in base al valore della `version` i risultati cambieranno notevolmente.

Di seguito verrà esposta la modalità di esecuzione e gestione del device In principio:

- `bandit.eval()` mette il modello in modalità di valutazione, disattivando dropout/batchnorm in training mode. Questo è appropriato per ottenere predizioni deterministiche dal modello durante la valutazione.
- `X = X.to(device)` e `y = y.to(device)` assicurano che i tensori siano collocati sullo stesso device del modello, evitando errori di tipo/destinazione durante le operazioni successive.
- Vengono inizializzate due variabili numeriche: `max_regret` e `cumulative_regret`, che saranno popolate e restituite come risultato della funzione.

**Operazioni principali nella branch `version == 1`**

1. `rewards_pred = bandit(X)`: il modello produce la matrice delle ricompense predette di forma $(N, A)$.
2. `max_rewards = torch.max(rewards_pred, dim=1)[0]`: per ogni riga (contesto) si calcola il valore massimo predetto tra le azioni. Risultato atteso $\in \mathbb{R}^N$.
3. `sampled_rewards = bandit.sample_rewards(X)`: si invoca il metodo di sampling già visto nel codice 3.4.3.

4. `actions = torch.argmax(sampled_rewards, dim=1)`: si seleziona l'azione scelta dalla politica del bandit in quel campione (argmax sulle ricompense campionate) — vettore di dimensione $N$ contenente indici interi.

5. `actual_rewards = compute_reward(y,k).to(device)`: viene calcolata la ricompensa *osservata* a partire dai target reali `y` e dal parametro $k$. Il risultato è traslato sul device corrente. Nel codice della branch 1 questa variabile viene calcolata ma non viene utilizzata per determinare `max_regret` o `cumulative_regret` (vedi osservazione sotto).

6. `selected_rewards = rewards_pred[torch.arange(X.size(0)), actions]`: per ciascun contesto si estrae il valore di ricompensa predetta corrispondente all'azione scelta dalla policy (indexing per riga).

7. `max_regret = max_rewards.sum().item()` e `cumulative_regret = (max_rewards - selected_rewards).sum().item()`: si sommano rispettivamente i massimi predetti su tutte le istanze e la differenza tra il massimo predetto e la ricompensa predetta per l'azione selezionata; il risultato viene convertito in valore Python scalare tramite `.item()`.

**Operazioni principali nella branch `version == 2`**

1. Si calcolano `rewards_pred = bandit(X)` come sopra.

2. `actual_rewards = compute_reward(y, k).to(device)`: qui le ricompense osservate, calcolate da `compute_reward`, vengono impiegate direttamente come benchmark. Il codice assegna `max_rewards = actual_rewards`, assumendo che `actual_rewards` rappresenti il valore massimo raggiungibile (o la ricompensa "ottima" osservata) per ciascun contesto.

3. `sampled_rewards = bandit.sample_rewards(X)` e `actions = torch.argmax(sampled_rewards, dim=1)`: come nella versione 1 si ottengono le azioni scelte in funzione di un campionamento.

4. `selected_rewards = rewards_pred[torch.arange(X.size(0)), actions]`: si estrae la predizione corrispondente all'azione scelta.

5. `max_regret` e `cumulative_regret` sono poi calcolati usando `actual_rewards` come riferimento di massima e la differenza rispetto a `selected_rewards`.

**Osservazioni rilevanti sul comportamento e sulle assunzioni del codice**

- **Diversa definizione di benchmark:** la differenza sostanziale tra le due branch è che la versione 1 usa come benchmark il *valore massimo predetto dal modello* per ogni contesto, mentre la versione 2 usa le *ricompense osservate* (derivate da `y`) come benchmark. Questa scelta cambia radicalmente la definizione operativa di `max_rewards` e quindi del calcolo del regret.

- **Shape e compatibilità:** l'accesso `rewards_pred[torch.arange(X.size(0)), actions]` presuppone che `rewards_pred` abbia shape $(N, A)$ e che `actions` sia un vettore di indici interi (dtype `torch.long`) di lunghezza $N$. Se `actions` o le dimensioni non combaciano, si verificheranno errori di indexing.

- **Ruolo di `bandit.sample_rewards`:** questa funzione è concettualmente cruciale: essa realizza il campionamento della politica (ad es. Thompson sampling). La sua implementazione determina il comportamento esplorativo del bandit e la natura statistica delle azioni selezionate; il codice assume che il campionamento sia riproducibile e compatibile con `torch.no_grad()`.

- **Possibilità di regret negativo:** il calcolo $\sum$(`max_rewards` − `selected_rewards`) può produrre valori negativi se `selected_rewards` supera `max_rewards` (ad esempio per incongruenze tra predizioni e benchmark). Ciò avviene in particolare se in versione 1 i massimi predetti sono in realtà inferiori alle predizioni per azioni specifiche, o per discrepanze di scala tra `actual_rewards` e `rewards_pred`. Il fenomeno è sintomatico di differenze semantiche tra "massimo predetto" e "ricompensa osservata".

- **Uso di `.sum().item()`:** la conversione a scalare Python è comoda per la reportistica, ma occorre attenzione in presenza di dataset molto grandi (possibili overflow) o quando si desidera conservare la granularità vettoriale delle misure.

- **Dipendenza da variabili esterne:** la presenza della variabile `version` (non passata come argomento) rende la funzione dipendente dallo stato esterno; questo può complicare il testing unitario e la riproducibilità se la variabile non è gestita esplicitamente.

- **Tipi e dispositivi:** il codice converte `actual_rewards` e i tensori su `device`, ma non verifica esplicitamente i `dtype` (es. `float32` vs `float64`) né la congruenza tra i tipi delle variabili indicizzanti. Tali discrepanze possono causare warning o errori silenti nelle operazioni aritmetiche.

**Output** La funzione restituisce due scalari Python:

(`max_regret`, `cumulative_regret`)

dove `max_regret` è la somma dei valori di riferimento (massimi) su tutte le istanze e `cumulative_regret` è la somma delle differenze per istanza tra il riferimento e la ricompensa effettivamente associata all'azione scelta dalla politica. Entrambe le quantità sono computate secondo la definizione concreta scelta nella branch corrispondente alla variabile `version`.

La descrizione sopra fornisce un resoconto tecnico e puntuale delle operazioni svolte dal frammento di codice; non vengono qui tratte conclusioni in merito alla capacità di generalizzazione del metodo, lasciando tale valutazione alle analisi empiriche e statistiche riportate separatamente.

```
def calculate_regret(bandit, X, y,k,device):
    bandit.eval()
    X = X.to(device)
    y = y.to(device)
    max_regret = 0
    cumulative_regret = 0
    if version == 1:
        with torch.no_grad():
            rewards_pred = bandit(X)
```

```
            max_rewards = torch.max(rewards_pred, dim=1)[0]
            sampled_rewards = bandit.sample_rewards(X)
            actions = torch.argmax(sampled_rewards, dim=1)
            actual_rewards = compute_reward(y,k).to(device)
            selected_rewards = rewards_pred[torch.arange(X.size(0)),
                actions]
            max_regret = max_rewards.sum().item()
            cumulative_regret = (max_rewards -
                selected_rewards).sum().item()
    elif version == 2:
        with torch.no_grad():
            rewards_pred = bandit(X)
            actual_rewards = compute_reward(y, k).to(device)
            max_rewards = actual_rewards  # Usa ricompense reali come
                massimo
            sampled_rewards = bandit.sample_rewards(X)
            actions = torch.argmax(sampled_rewards, dim=1)
            selected_rewards = rewards_pred[torch.arange(X.size(0)),
                actions]
            max_regret = max_rewards.sum().item()
            cumulative_regret = (max_rewards -
                selected_rewards).sum().item()

    return max_regret, cumulative_regret
```

**Ricerca del Contesto Ottimale** La funzione `find_optimal_input` identifica il contesto e l'azione che massimizzano la ricompensa attesa, ritrasformando le feature standardizzate (`scaler_features.inverse_transform`) per ottenere valori interpretabili. Utilizza l'MLP per stimare vendite, spesa e click per il contesto ottimale:

```
def find_optimal_input(bandit, mlp_model, X_test, y_test,
    scaler_features, features, k, device):
    bandit.eval()
    X_test = X_test.to(device)

    with torch.no_grad():
        rewards_pred = bandit(X_test)
        max_rewards, best_actions = torch.max(rewards_pred, dim=1)
        best_idx = torch.argmax(max_rewards)
        best_context = X_test[best_idx].cpu().numpy()
        best_action = best_actions[best_idx].item()
        best_reward = max_rewards[best_idx].item()

        non_sbert_indices = [features.index(f) for f in
            ['Impressions', 'lancio', 'performance', 'Day_of_Week',
            'Match Type Encoded', 'similarity', 'Cluster']]
        best_context_real = best_context.copy()
        best_context_real[non_sbert_indices] =
            scaler_features.inverse_transform(
             best_context[non_sbert_indices].reshape(1, -1))[0]

        best_context_tensor = torch.tensor(best_context,
            dtype=torch.float32).unsqueeze(0).to(device)
        y_pred = mlp_model(best_context_tensor)
        y_pred_real = np.expm1(y_pred.cpu().numpy())[0]
        sales = y_pred_real[2]
        spend = y_pred_real[1]
```

```
        click = y_pred_real[0]

    return {
        'best_context': best_context_real,
        'best_action': best_action,
        'best_reward': best_reward,
        'predicted_sales': sales,
        'predicted_spend': spend,
        'predicted_click': click,
        'features': features
    }
```

### Dettagli della Pipeline

La pipeline completa integra le seguenti fasi:

1. **Pre-elaborazione dei dati**: I dati sono stati preprocessati come descritto nel Capitolo 2.2, includendo la codifica delle feature categoriali, la standardizzazione delle feature non-SBERT, la trasformazione logaritmica dei target e la generazione della feature `Cluster` tramite UMAP e HDBSCAN. I dati sono stati convertiti in tensori PyTorch e suddivisi in training, validation e test.

2. **Inizializzazione del bandit**: La funzione `initialize_bandit_with_mlp` utilizza le predizioni dell'MLP per calcolare le ricompense iniziali e inizializzare `mu` e `sigma` per ogni azione, basandosi sui contesti con ricompense elevate (quantile 0.8).

3. **Addestramento**: La funzione `train_bandit` addestra il modello per un numero specificato di epoche, utilizzando batch per aggiornare i modelli lineari e le distribuzioni gaussiane. La perdita MSE è minimizzata per allineare le ricompense previste a quelle osservate, con un approccio Thompson Sampling per selezionare le azioni.

4. **Valutazione del regret**: La funzione `calculate_regret` valuta le prestazioni calcolando il *max_regret* e il *cumulative_regret*, fornendo una misura quantitativa dell'efficacia del modello.

5. **Ottimizzazione del contesto**: La funzione `find_optimal_input` identifica il contesto e l'azione ottimali, ritrasformando le feature per interpretabilità e utilizzando l'MLP per stimare i target associati.

La pipeline è progettata per essere modulare e scalabile, con salvataggio delle stime delle ricompense e delle azioni in `rewards_history` e `actions_history` per analisi successive.

### Iperparametri Testati

Gli iperparametri del bandit includono:

- `num_actions`: Numero di azioni possibili, nel codice viene inizializzata a 5, fondamentale per definire i bracci.
- `context_dim`: Dimensione del contesto (es. 391 con `Cluster`).
- `lr`: Tasso di apprendimento per l'ottimizzatore Adam, passato come parametro alla funzione `train_bandit`.

- `batch_size`: Dimensione del batch per l'addestramento.
- `epochs`: Numero di epoche per l'addestramento.
- `k = 0.5`: Fattore di penalizzazione nella funzione di ricompensa.

È stata condotta una *grid search*, i parametri sono stati scelti per bilanciare convergenza e prestazioni.

**Fonti**

Il modello si basa su Thompson Sampling [49], con dettagli implementativi ispirati a [11]. La teoria dei *Contextual Bandit* è descritta in [33] e [32]. L'inizializzazione *warm-start* è supportata da [22]. L'implementazione in PyTorch si basa su [39].

### 3.4.4 Altri esperimenti effettuati

In questa sezione vengono presentati e discussi gli esperimenti condotti al fine di valutare l'efficacia del modello di *Contextual Bandit* applicato alle campagne pubblicitarie PPC di Amazon. L'obiettivo principale è stato quello di analizzare in che modo l'integrazione di una rete *Multi-Layer Perceptron* (MLP) potesse contribuire a ridurre il tempo necessario al bandit per convergere verso un comportamento ottimale, limitando così il *regret* nelle fasi iniziali di apprendimento. In scenari caratterizzati da scarsità di dati storici, come quello considerato in questa tesi, l'utilizzo di un modello neurale permette infatti di realizzare un processo di *warm start*, evitando di partire completamente da zero nella fase esplorativa.

Durante le prime iterazioni sperimentali è emerso un problema critico: il *regret* calcolato dal bandit risultava negativo e di magnitudine elevata (in media alcune centinaia), con valori che in alcuni casi raggiungevano circa $-800\%$. Una tale situazione non è matematicamente plausibile, in quanto implica che l'algoritmo sia in grado di performare meglio rispetto allo scenario reale di riferimento, portando a risultati distorti e potenzialmente fuorvianti. Tale anomalia era riconducibile alla modalità con cui era stato definito il meccanismo di calcolo della ricompensa (*reward*). Nella prima implementazione (3.4.3), infatti, la reward era derivata direttamente dai dati reali ($y$), con la conseguenza che il bandit poteva erroneamente sovrastimare le vendite. Questo effetto era ulteriormente amplificato dall'ottimizzazione del parametro $K$, che modulava le predizioni verso valori atipici, causando rewards artificialmente elevate.

Per superare questa criticità, si è modificata la definizione della funzione di ricompensa, calcolandola non più sui dati reali, bensì sulle predizioni generate dal bandit stesso, in linea con l'impostazione classica della teoria dei bandit. In questo modo, i risultati ottenuti si sono dimostrati più coerenti con la realtà: il regret medio è tornato ad assumere valori positivi, oscillando in un intervallo compreso tra lo 0% e il 40%. Tale correzione ha reso i risultati più interpretabili e scientificamente validi.

Oltre alla ridefinizione della funzione di ricompensa, è stata condotta una serie di esperimenti volti a confrontare diverse configurazioni del modello. In particolare, si è analizzato:

- l'utilizzo congiunto di bandit e MLP(3.4.3);
- l'impiego del bandit in assenza di MLP;

- l'integrazione di un'MLP priva di pesi iniziali, per valutare l'impatto dell'addestramento da zero;
- l'inserimento di una rete neurale di dimensioni ridotte all'interno del bandit, al fine di esplorare l'efficacia di architetture più compatte.

È opportuno sottolineare che i primi tre esperimenti sopra menzionati non verranno descritti nel dettaglio dal punto di vista implementativo, poiché la metodologia adottata risulta relativamente banale. In particolare, nel secondo caso è stato sufficiente disattivare la riga di codice responsabile della chiamata alla MLP, consentendo così al bandit di operare senza alcun supporto neurale. Analogamente, nel terzo esperimento è stata disattivata la riga che caricava i pesi pre-addestrati della rete, forzando di fatto l'MLP a partire da zero. Tali modifiche, pur essendo concettualmente rilevanti per l'analisi comparativa dei risultati, non introducono alcuna complessità tecnica significativa e non richiedono pertanto un'illustrazione programmatica approfondita all'interno della tesi.

### Utilizzo di una Neural Network nel contextual bandit

In questa implementazione del `ContextualBandit` si introduce la possibilità di utilizzare, per ciascuna azione, una rete neurale di tipo *Multi-Layer Perceptron* (MLP) anziché un singolo strato lineare. Questo consente al modello di apprendere rappresentazioni più complesse del contesto, migliorando potenzialmente la capacità predittiva in scenari caratterizzati da alta dimensionalità e non linearità. Di seguito si riporta il codice:

```
class ContextualBandit(nn.Module):
    def __init__(self, context_dim, num_actions):
        super(ContextualBandit, self).__init__()
        self.num_actions = num_actions
        self.context_dim = context_dim

        if(bandit_nn):
            self.action_models = nn.ModuleList([
            nn.Sequential(
                nn.Linear(context_dim, 64),  # Primo strato nascosto
                nn.ReLU(),
                nn.Linear(64, 32),           # Secondo strato nascosto
                nn.ReLU(),
                nn.Linear(32, 1)             # Strato di output
            ) for _ in range(num_actions)
            ])
        else:
            self.action_models = nn.ModuleList([nn.Linear(context_dim,
                1) for _ in range(num_actions)])

        self.mu = torch.zeros(num_actions, context_dim)
        self.sigma = torch.ones(num_actions, context_dim)
```

Il cuore di questa implementazione risiede nel blocco condizionato dalla variabile `bandit_nn`. Quando essa è attiva, ciascun modello associato a una delle possibili azioni viene realizzato come una rete neurale sequenziale composta da tre livelli principali:

- **Primo strato nascosto:** `nn.Linear(context_dim, 64)`. Questo livello riceve in input il contesto, rappresentato da un vettore di dimensione `context_dim`, e lo proietta in uno spazio latente di dimensione 64. La scelta di aumentare la

dimensionalità rispetto all'input consente al modello di catturare combinazioni più ricche e complesse delle variabili contestuali. L'attivazione `ReLU` introdotta subito dopo riduce la linearità del modello e permette di apprendere funzioni non lineari, fondamentali in scenari reali dove le relazioni tra contesto e reward non sono quasi mai lineari.

- **Secondo strato nascosto:** `nn.Linear(64, 32)`. Qui si riduce la dimensionalità dallo spazio a 64 unità a 32 unità. Questa riduzione graduale ha lo scopo di comprimere le informazioni mantenendo al contempo le caratteristiche più rilevanti. Anche in questo caso, la funzione di attivazione `ReLU` permette di introdurre ulteriore non linearità e di garantire una migliore capacità di generalizzazione.

- **Strato di output:** `nn.Linear(32, 1)`. L'ultimo livello restituisce un singolo valore, che rappresenta la stima del reward atteso per l'azione corrispondente dato il contesto. Si tratta di un'uscita scalare, coerente con il paradigma del bandit, dove l'obiettivo è stimare il guadagno potenziale di ciascuna azione in funzione delle informazioni disponibili.

Nel caso in cui la variabile `bandit_nn` non sia attiva, il modello adotta un approccio più semplice e lineare: per ciascuna azione viene definito un singolo strato `nn.Linear(context_dim, 1)`. Questa configurazione corrisponde a un modello lineare classico(già visto nel paragrafo 3.4.3) , in cui la previsione del reward è ottenuta tramite una combinazione lineare delle feature contestuali.

Questa implementazione risulta dunque flessibile: da un lato permette di mantenere la semplicità e l'efficienza di un modello lineare, dall'altro consente di sfruttare la potenza espressiva delle reti neurali profonde qualora si voglia affrontare un problema con relazioni complesse tra contesto e reward.

### Optimal inputs: ricerca delle migliori keyword

Un ulteriore elemento di rilievo negli esperimenti effettuati è rappresentato dalla funzione `find_optimal_input`, il cui obiettivo è individuare, a partire dai modelli già allenati, le combinazioni ottimali di iperparametri di addestramento per trovare il bandit che meglio minimizza il regret. Questa procedura permette quindi di ricavare, per la configurazione scelta, le 10 migliori azioni in termini di reward previsto, fornendo una base operativa utile per la valutazione dell'impatto delle scelte di targeting sulle performance delle campagne PPC.

Il codice della funzione è riportato di seguito:

```
def find_optimal_input(bandit, mlp_model, X_test, y_test,
    scaler_features, features, k, device, top_n=10):
     bandit.eval()
     X_test = X_test.to(device)

     with torch.no_grad():
         # Predict rewards for each context and action
         rewards_pred = bandit(X_test)  # Shape: (N, num_actions)
         max_rewards, best_actions = torch.max(rewards_pred, dim=1)  #
            Max reward and action per context

         # Get the top N contexts with highest rewards
         top_n_indices = torch.topk(max_rewards, k=min(top_n,
            len(max_rewards)), largest=True).indices
```

```
        top_contexts = X_test[top_n_indices].cpu().numpy()
        top_actions = best_actions[top_n_indices].cpu().numpy()
        top_rewards = max_rewards[top_n_indices].cpu().numpy()

        # Convert contexts to real values (inverse standardization)
        non_sbert_indices = [features.index(f) for f in ['lancio',
            'performance', 'Day_of_Week', 'Match Type Encoded',
            'similarity']]
        top_contexts_real = top_contexts.copy()
        top_contexts_real[:, non_sbert_indices] =
            scaler_features.inverse_transform(top_contexts[:,
            non_sbert_indices])

        # Predict targets with MLP for Sales, Spend, Clicks, and
            Impressions
        if(mlp_model):
            top_contexts_tensor = torch.tensor(top_contexts,
                dtype=torch.float32).to(device)
            y_pred = mlp_model(top_contexts_tensor)
            y_pred_real = np.expm1(y_pred.cpu().numpy())
        else:
            y_pred_real = np.expm1(y_test[top_n_indices].cpu().numpy())

        results = []
        for i in range(len(top_n_indices)):
            result = {
                'best_context': top_contexts_real[i],
                'best_action': top_actions[i],
                'best_reward': top_rewards[i],
                'predicted_sales': y_pred_real[i, 2],  # 7 Day Total
                    Sales
                'predicted_spend': y_pred_real[i, 1],  # Spend
                'predicted_click': y_pred_real[i, 0],  # Clicks
                'predicted_impression': y_pred_real[i, 5],  #
                    Impressions
                'features': features
            }
            results.append(result)

    return results
```

La logica implementata si articola nei seguenti passaggi principali:

- **Valutazione del bandit:** il modello viene posto in modalità `eval` e, senza aggiornamento dei gradienti, calcola i reward previsti per ciascun contesto e azione.
- **Selezione delle azioni ottimali:** per ogni contesto si individua l'azione con reward massimo e si ordinano le migliori combinazioni, estraendo i primi $N$ risultati (per default $N = 10$).
- **Riconversione delle feature:** i contesti selezionati vengono riportati alle scale originali tramite l'inversa dello standardizzatore, così da rendere leggibili i valori reali di input.
- **Predizione delle metriche di business:** se disponibile, la rete MLP viene utilizzata per stimare grandezze operative (vendite, spesa, click, impression). In assenza di MLP, vengono invece recuperati i valori reali di test.

- **Strutturazione dei risultati:** per ciascuna delle top combinazioni viene creato un dizionario che include contesto, azione ottimale, reward previsto e metriche di business associate.

In sintesi, la funzione consente di tradurre le capacità predittive del modello in raccomandazioni pratiche, identificando i contesti più promettenti e le azioni più redditizie da intraprendere, e rendendo così il bandit uno strumento operativo per l'ottimizzazione delle campagne pubblicitarie.

**Tabella 3.1:** Esempio di "optimal input": Top 10 combinazioni ottimali di keyword e parametri stimati dal bandit con MLP.

| Keyword | Comb. | Azione | Reward Previsto | Vendite Previste | Spesa Prevista | Click Previsti |
|---|---|---|---|---|---|---|
| burro struccante da viaggio | 1 | 3 | 0.1150 | 0.87 | 1.00 | 13.00 |
| burro struccante al cocco | 2 | 3 | 0.1141 | 1.37 | 1.00 | 1.00 |
| detergente viso | 3 | 3 | 0.1113 | 0.46 | 1.00 | 4.00 |
| latte detergente viso pelle mista | 4 | 3 | 0.1102 | 1.53 | 1.00 | 1.00 |
| burro struccante viso e occhi naturale | 5 | 3 | 0.1098 | 0.42 | 1.00 | 3.00 |
| burro struccante viso e occhi | 6 | 3 | 0.1098 | 2.04 | 3.00 | 1255.00 |
| struccante viso e occhi | 7 | 3 | 0.1098 | 1.22 | 2.00 | 54.00 |
| garnier struccante waterproof | 8 | 3 | 0.1097 | 0.91 | 1.00 | 1.00 |
| struccante viso e occhi | 9 | 3 | 0.1096 | 0.42 | 1.00 | 31.00 |
| burro struccante spugna | 10 | 3 | 0.1092 | 0.48 | 1.00 | 1.00 |

Oltre alle colonne visibili in Tabella 3.1 sono presenti nell'ouput originale del modello anche ulteriori colonne quali `Predicted_Impression`,`lancio`,`performance`,`Day_of_Week`,`Match Type Encoded`,`similarity`,`sbert_0`,`sbert_1`,...,`sbert_383` i quali sono stati esclusi per permettere una migliore visualizzazione di un esempio di output.

# Capitolo 4

# Risultati ottenuti

In questo capitolo vengono presentati e analizzati i risultati sperimentali derivanti dai modelli introdotti nel Capitolo 3. L'obiettivo principale è quello di valutare le prestazioni delle diverse configurazioni proposte, discutendo in maniera critica gli errori di predizione e i valori di *regret* associati. Per ciascun modello testato, e per le relative varianti, verranno quindi messi in evidenza i punti di forza, le limitazioni e le implicazioni in termini di applicabilità pratica al contesto delle campagne pubblicitarie PPC su Amazon.

Un'attenzione particolare sarà riservata alle architetture che integrano reti neurali, con particolare focus sull'approccio *Contextual Bandit*. Quest'ultimo rappresenta infatti il nucleo innovativo della ricerca, ed è il modello che meglio si presta a mettere in evidenza la relazione tra contesto e scelte di ottimizzazione pubblicitaria. Nei casi più significativi, i risultati saranno accompagnati da rappresentazioni grafiche costruite ad hoc, al fine di facilitare la comprensione visiva dei comportamenti osservati e delle dinamiche di apprendimento.

La struttura del capitolo segue un approccio comparativo: dapprima verranno presentati i risultati dei modelli di base, successivamente quelli delle loro varianti, per poi approfondire i casi in cui l'integrazione di una MLP o l'applicazione del *Contextual Bandit* hanno portato a un miglioramento (o peggioramento) delle prestazioni. In questo modo, sarà possibile costruire un quadro completo che metta in luce l'efficacia e le criticità delle diverse soluzioni esplorate.

## 4.1 Analisi dei Risultati dei Modelli di Regressione

I risultati riportati nella Tabella 4.1 descrivono le performance di tre modelli di apprendimento supervisionato — *XGBoost*, *LGBMRegressor* e *RandomForestRegressor* — nella predizione di cinque metriche chiave delle campagne pubblicitarie Amazon: `Clicks`, `Spend`, `7 Day Total Sales`, `7 Day Total Orders (#)` e `7 Day Conversion Rate`. Le metriche di valutazione considerate sono l'Errore Assoluto Medio (MAE), l'Errore Quadratico Medio della Radice (RMSE) e il coefficiente di determinazione ($R^2$).

Di seguito si presenta un'analisi dettagliata delle performance dei modelli, confrontando punti di forza e limitazioni per ciascun target, con particolare attenzione alla loro applicabilità nel contesto dell'ottimizzazione delle campagne pubblicitarie.

**Tabella 4.1:** Sintesi delle metriche di performance per modello e target (esclusi: *Timestamp*, *Impressions* e colonna *MAPE*). I valori migliori di $R^2$ per ciascun target sono evidenziati in grassetto.

| **Modello** | **Target** | **MAE** | **RMSE** | $R^2$ |
|---|---|---|---|---|
| XGBoost | Clicks | 0.1051 | 0.2441 | 0.5527 |
| | Spend | 0.2718 | 0.4175 | 0.5362 |
| | 7 Day Total Sales | 1.4191 | 3.5152 | **0.5241** |
| | 7 Day Total Orders (#) | 0.0734 | 0.1723 | 0.4903 |
| | 7 Day Conversion Rate | 0.0627 | 0.1512 | 0.5242 |
| LGBMRegressor | Clicks | 0.3577 | 0.3812 | -6.9549 |
| | Spend | 0.3070 | 0.3644 | -1.0453 |
| | 7 Day Total Sales | 0.4037 | 1.7457 | -5.1210 |
| | 7 Day Total Orders (#) | 0.0440 | 0.1084 | **0.5698** |
| | 7 Day Conversion Rate | 0.0399 | 0.0992 | **0.5790** |
| RandomForestRegressor | Clicks | 0.0579 | 0.1748 | **0.7707** |
| | Spend | 0.2131 | 0.3401 | **0.6923** |
| | 7 Day Total Sales | 1.7070 | 4.1251 | 0.3447 |
| | 7 Day Total Orders (#) | 0.0835 | 0.1955 | 0.3439 |
| | 7 Day Conversion Rate | 0.0781 | 0.1827 | 0.3056 |

### 4.1.1 Performance sui Clicks

Per la metrica `Clicks`, il *RandomForestRegressor* ottiene le migliori performance, con un MAE di 0.0579, un RMSE di 0.1748 e un $R^2$ pari a 0.7707, indicando un'elevata capacità predittiva e una buona spiegazione della varianza nei dati. *XGBoost* segue con un MAE di 0.1051, un RMSE di 0.2441 e un $R^2$ di 0.5527, mostrando prestazioni accettabili ma inferiori rispetto a RandomForest. *LGBMRegressor*, invece, presenta risultati insoddisfacenti, con un MAE di 0.3577, un RMSE di 0.3812 e un $R^2$ negativo (-6.9549), suggerendo che il modello non riesca a catturare la variabilità dei dati e produca predizioni meno accurate della media. Questo risultato può derivare da una configurazione iperparametrica non ottimale o da una sensibilità del modello alla distribuzione dei dati.

### 4.1.2 Performance su Spend

Per `Spend`, il *RandomForestRegressor* conferma la propria superiorità, con un MAE di 0.2131, un RMSE di 0.3401 e un $R^2$ di 0.6923, dimostrando una buona capacità di predire i costi delle campagne. *XGBoost* mostra risultati simili, con un MAE di 0.2718, un RMSE di 0.4175 e un $R^2$ di 0.5362, leggermente inferiori ma comunque competitivi. *LGBMRegressor*, pur presentando un MAE (0.3070) e un RMSE (0.3644) comparabili, mostra un $R^2$ negativo (-1.0453), segnalando una scarsa capacità di spiegare la varianza rispetto a un modello baseline. La superiorità di RandomForest può essere attribuita alla sua capacità di modellare relazioni non lineari, mentre LGBM sembra soffrire di una configurazione non adeguata per questa metrica.

### 4.1.3 Performance su 7 Day Total Sales

Per `7 Day Total Sales`, *LGBMRegressor* ottiene il MAE più basso (0.4037), suggerendo una maggiore precisione nelle predizioni puntuali rispetto a *XGBoost* (MAE 1.4191) e *RandomForest* (MAE 1.7070). Tuttavia, il suo RMSE (1.7457) e il $R^2$ negativo (-5.1210) evidenziano errori significativi e un'incapacità di catturare la variabilità complessiva. *XGBoost* (RMSE 3.5152, $R^2$ 0.5241) e *RandomForest* (RMSE 4.1251, $R^2$ 0.3447) mostrano performance migliori in termini di $R^2$, ma errori puntuali più elevati. La difficoltà di predire le vendite totali, influenzate da fattori esterni come stagionalità e promozioni, potrebbe spiegare le basse performance complessive.

### 4.1.4 Performance su 7 Day Total Orders (#)

Per `7 Day Total Orders (#)`, *LGBMRegressor* si distingue con il MAE più basso (0.0440), un RMSE di 0.1084 e un $R^2$ pari a 0.5698, dimostrando una buona capacità predittiva. *XGBoost* segue con un MAE di 0.0734, un RMSE di 0.1723 e un $R^2$ di 0.4903, mentre *RandomForest* risulta meno performante (MAE 0.0835, RMSE 0.1955, $R^2$ 0.3439). La solidità di LGBM su questa metrica può essere attribuita alla sua capacità di gestire distribuzioni complesse grazie al boosting basato su gradienti.

### 4.1.5 Performance su 7 Day Conversion Rate

Per `7 Day Conversion Rate`, *LGBMRegressor* raggiunge nuovamente le migliori performance, con un MAE di 0.0399, un RMSE di 0.0992 e un $R^2$ pari a 0.5790, superando *XGBoost* (MAE 0.0627, RMSE 0.1512, $R^2$ 0.5242) e *RandomForest* (MAE 0.0781, RMSE

0.1827, $R^2$ 0.3056). Ciò indica che LGBM è particolarmente efficace nel predire il tasso di conversione, una metrica cruciale per valutare l'efficacia delle campagne pubblicitarie. Le performance inferiori di RandomForest possono derivare dalla tendenza a sovradattarsi in presenza di relazioni complesse, mentre XGBoost fornisce un buon compromesso tra accuratezza e generalizzazione.

### 4.1.6 Confronto Complessivo e Riflessioni

Analizzando le performance complessive, il *RandomForestRegressor* emerge come il più robusto su `Clicks` e `Spend`, con i valori più alti di $R^2$ (0.7707 e 0.6923). Per contro, su `7 Day Total Sales`, `7 Day Total Orders (#)` e `7 Day Conversion Rate`, *LGBMRegressor* si dimostra superiore in termini di MAE, pur mostrando valori di $R^2$ negativi su altre metriche. *XGBoost* rappresenta un compromesso equilibrato, offrendo prestazioni consistenti senza eccellere in alcun target specifico.

Le differenze prestazionali possono essere attribuite a peculiarità dei modelli:

- **RandomForestRegressor**: eccelle nella gestione di relazioni non lineari e si dimostra robusto su dataset rumorosi, ma presenta limiti su metriche più variabili e complessità computazionale elevata.
- **LGBMRegressor**: performa bene su target discreti come `Orders` e `Conversion Rate`, ma soffre su variabili continue come `Clicks`, `Spend` e `Sales`, probabilmente a causa di un tuning iperparametrico non ottimale.
- **XGBoost**: bilancia precisione e generalizzazione, ma senza raggiungere le migliori performance in alcuna metrica. La sua configurazione appare stabile, ma non pienamente ottimizzata.

### 4.1.7 Limitazioni e Prospettive Future

Le performance dei modelli evidenziano alcune limitazioni. I valori negativi di $R^2$ in *LGBMRegressor* per alcune metriche suggeriscono un potenziale sovradattamento o un'incapacità di catturare la variabilità intrinseca dei dati. Inoltre, le difficoltà riscontrate nella predizione di `7 Day Total Sales` mostrano la necessità di includere fattori esterni come stagionalità e promozioni.

Per migliorare le performance, si propongono:

- tuning più approfondito degli iperparametri, specialmente per *LGBMRegressor*;
- arricchimento delle feature con informazioni esterne (dati stagionali, prezzi, attività dei competitor);
- uso di modelli ensemble che combinino i punti di forza di RandomForest, XGBoost e LGBM.

### 4.1.8 Implicazioni per l'Ottimizzazione delle Campagne

I risultati ottenuti sono stati utilizzati per inizializzare il *contextual bandit*, come descritto nel capitolo successivo. La scelta di RandomForest per `Clicks` e `Spend`, e di LGBM per `7 Day Total Orders (#)` e `7 Day Conversion Rate`, si rivela strategica per fornire

predizioni iniziali più accurate, migliorando l'efficacia dell'inizializzazione e riducendo il regret cumulativo. Tuttavia, il bilanciamento tra accuratezza puntuale (MAE) e capacità di generalizzazione ($R^2$) resta cruciale per selezionare il modello più appropriato in base al target e al contesto operativo.

## 4.2 Analisi dei Risultati con la Rete Neurale MLP

Dopo aver esaminato i modelli di regressione basati su alberi decisionali, l'attenzione è stata rivolta alla valutazione di una rete neurale multilayer perceptron (MLP), addestrata sugli stessi dati utilizzati in precedenza. L'obiettivo era verificare se un approccio basato su modelli neurali potesse migliorare le capacità predittive, soprattutto in contesti caratterizzati da non linearità e interazioni complesse tra variabili.

Il modello MLP è stato configurato con più strati densi e funzioni di attivazione non lineari, addestrato variando iperparametri chiave quali il tipo di ottimizzatore, la dimensione del batch e il learning rate. Le metriche di valutazione adottate sono state analoghe a quelle dei modelli di regressione tradizionali, con un'attenzione particolare al valore medio dell'errore quadratico medio della radice (RMSE) calcolato per ciascun target.

### 4.2.1 Statistiche descrittive delle metriche RMSE

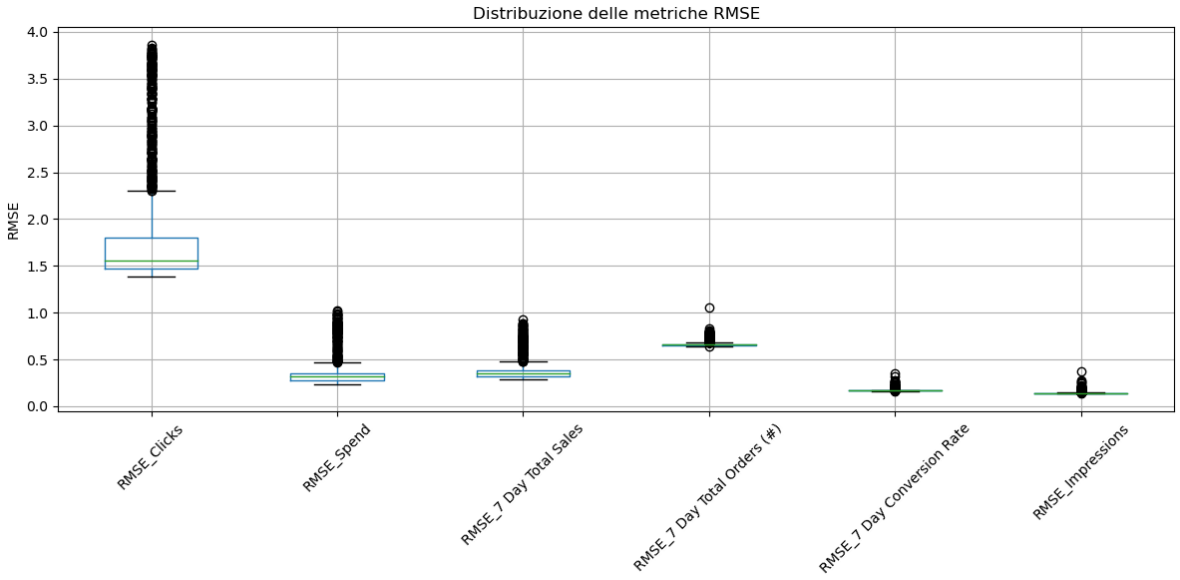


**Figura 4.1:** Distribuzione delle metriche RMSE

Le statistiche descrittive dei valori di RMSE per le diverse metriche (Fig. 4.1) mostrano una distribuzione relativamente stabile, con deviazioni standard contenute. In particolare, i valori medi oscillano tra 0.1408 (`Impressions`) e 1.7253 (`Clicks`). Si osserva inoltre che i valori minimi e massimi si collocano entro un intervallo ristretto, a conferma della stabilità del processo di addestramento.

Ad esempio, per la metrica `7 Day Total Orders (#)` il valore medio di RMSE è pari a 0.6639, con una deviazione standard di appena 0.0216, a testimonianza di una variabilità molto limitata. All'opposto, `Clicks` presenta un RMSE medio sensibilmente

più elevato (1.7253) e con maggiore variabilità (std = 0.4450), indicando una maggiore complessità predittiva.

Un grafico dedicato (Fig. 4.1) illustra in maniera visiva la distribuzione dei valori di RMSE per ciascun target, mettendo in evidenza le differenze di scala e di variabilità.

### 4.2.2 Matrice di correlazione tra metriche RMSE

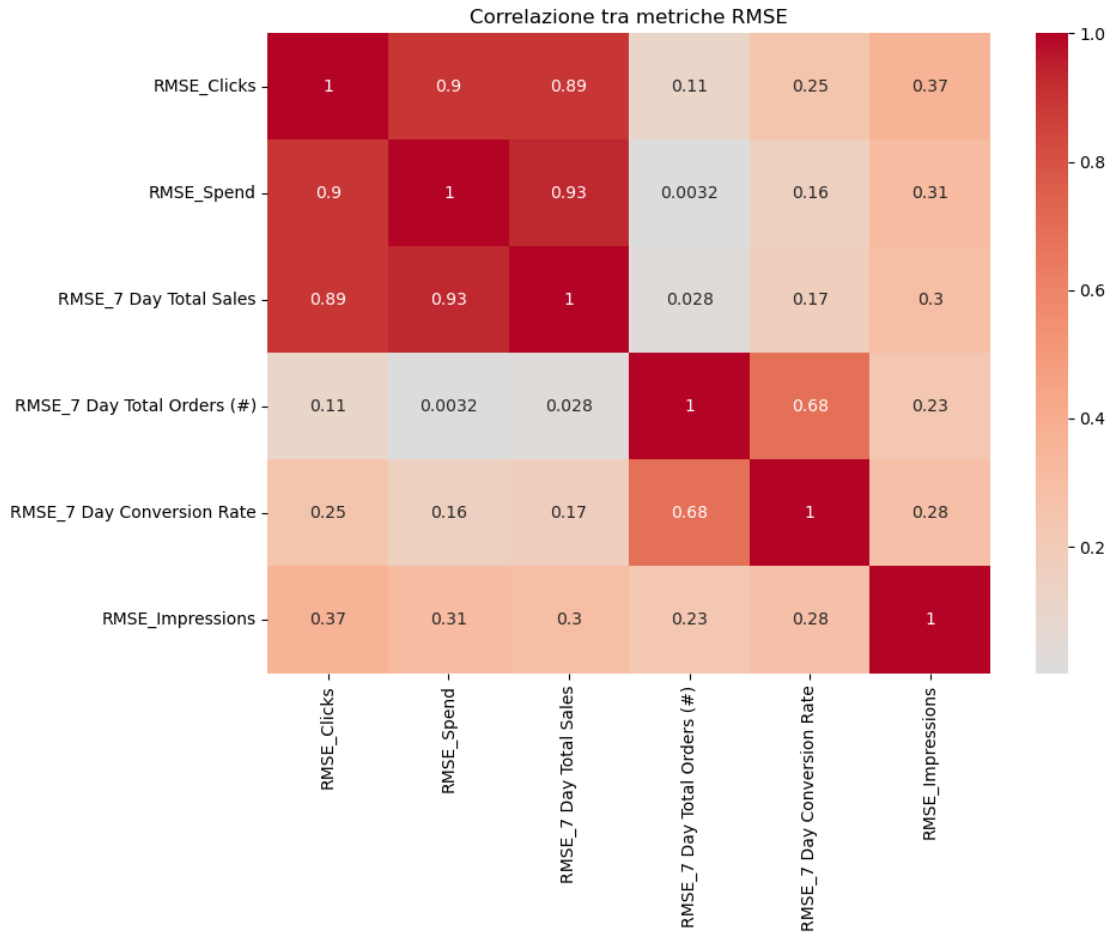


**Figura 4.2:** Correlazione metrica RMSE tra i target

La matrice di correlazione (4.2) calcolata sui valori di RMSE evidenzia relazioni interessanti tra le metriche. Si nota una forte correlazione tra `Clicks`, `Spend` e `7 Day Total Sales`, con coefficienti superiori a 0.89, suggerendo che gli errori commessi dal modello in queste metriche tendono a muoversi in maniera sincrona.

D'altra parte, la correlazione tra `7 Day Total Orders (#)` e le prime tre metriche è molto debole (coefficiente massimo 0.1081), indicando che la predizione del numero di ordini segue un comportamento indipendente rispetto alle metriche legate a click e spesa. Una correlazione particolarmente significativa emerge invece tra `7 Day Total Orders (#)` e `7 Day Conversion Rate` (0.6816), dato atteso in quanto entrambe le metriche derivano da relazioni dirette con le conversioni.

Il grafico associato alla matrice di correlazione (4.2) consente di visualizzare in modo intuitivo queste dipendenze, rafforzando l'interpretazione numerica.

### 4.2.3 Effetto del tipo di ottimizzatore

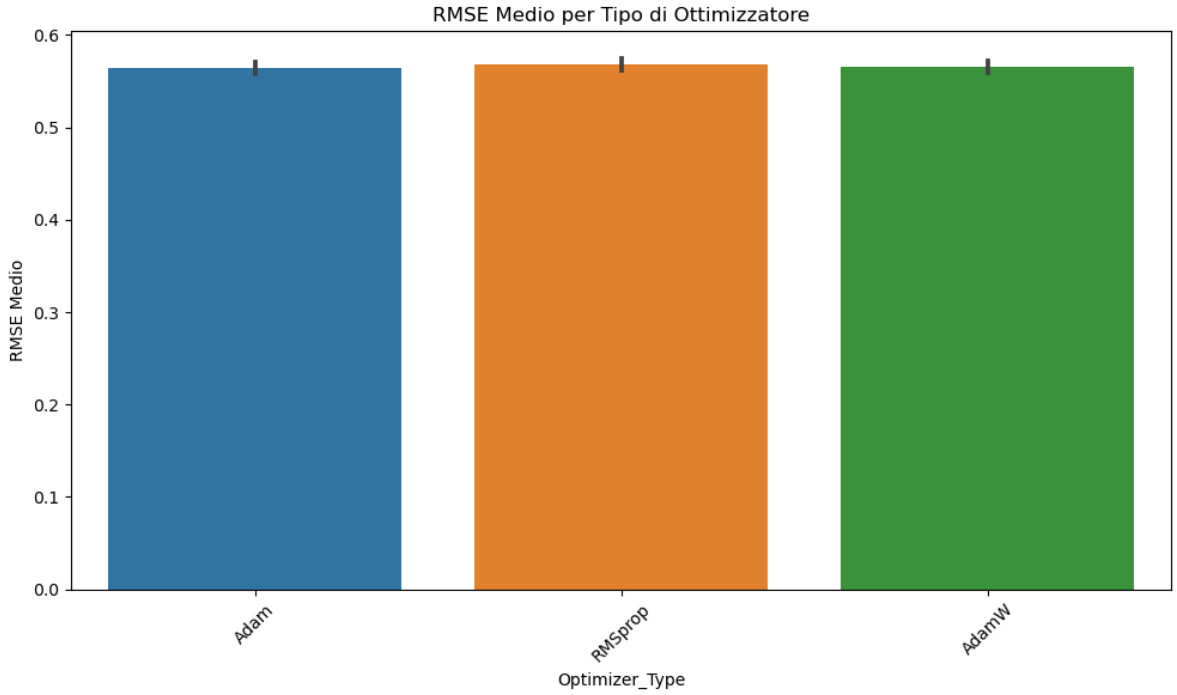


**Figura 4.3:** Confronto tra i vari optimizer e il loro RMSE medio

L'analisi degli ottimizzatori adottati (`Adam`, `AdamW` e `RMSprop`) (4.3) mostra performance sostanzialmente equivalenti. I valori medi di RMSE si collocano rispettivamente a 0.5645, 0.5655 e 0.5684, con differenze minime e deviazioni standard simili (intorno a 0.10). Questo risultato indica che, a parità di altre condizioni, il tipo di ottimizzatore non rappresenta un fattore discriminante per il miglioramento sostanziale delle performance.

Il grafico di confronto (4.3) evidenzia questa sovrapposizione, mostrando distribuzioni molto vicine tra i tre metodi.

### 4.2.4 Effetto della dimensione del batch

La dimensione del batch ha mostrato un impatto più rilevante sulle performance. Con un batch size di 256, l'RMSE medio si riduce a 0.5429, inferiore rispetto a 0.5649 per 512 e a 0.5907 per 1024. La crescita dell'errore con batch più ampi suggerisce che aggiornamenti più frequenti (batch più piccoli) favoriscono una convergenza più stabile e accurata.

Il grafico relativo (4.4) conferma tale andamento, evidenziando come le distribuzioni di RMSE si spostino progressivamente verso valori più elevati con l'aumento del batch size.

### 4.2.5 Influenza del learning rate

Il learning rate si conferma un iperparametro cruciale per l'ottimizzazione della rete neurale. Valori troppo bassi (ad esempio $10^{-5}$) producono RMSE medi più elevati, con casi che superano 0.77, indicando una convergenza lenta e poco efficace. Al contrario, un learning rate pari a $5 \times 10^{-4}$ porta a valori medi sensibilmente inferiori (fino a 0.4981

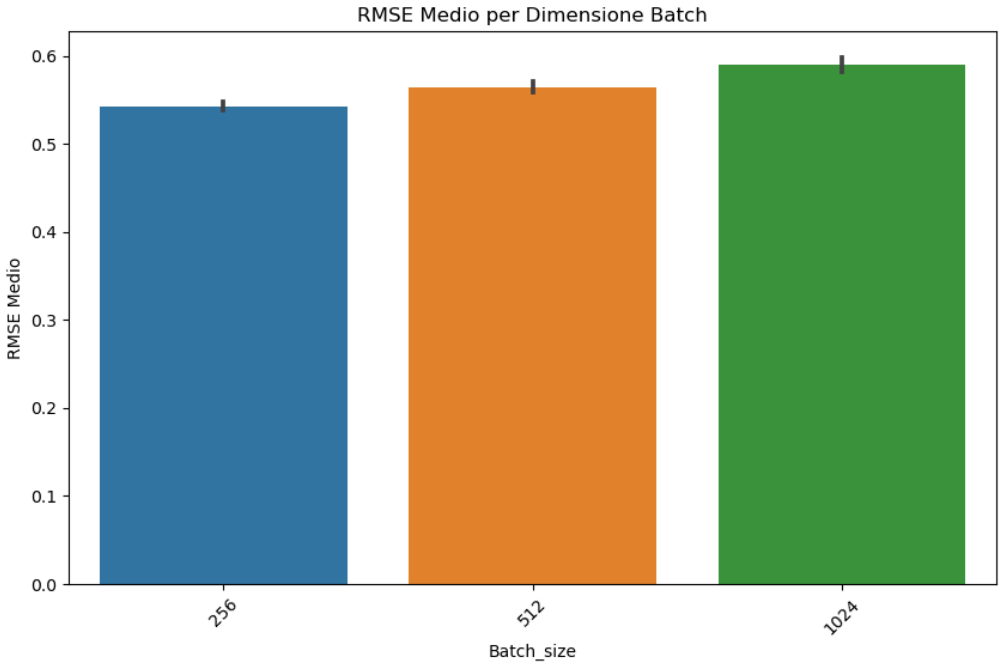


**Figura 4.4:** Confronto tra batch_size e il loro RMSE medio

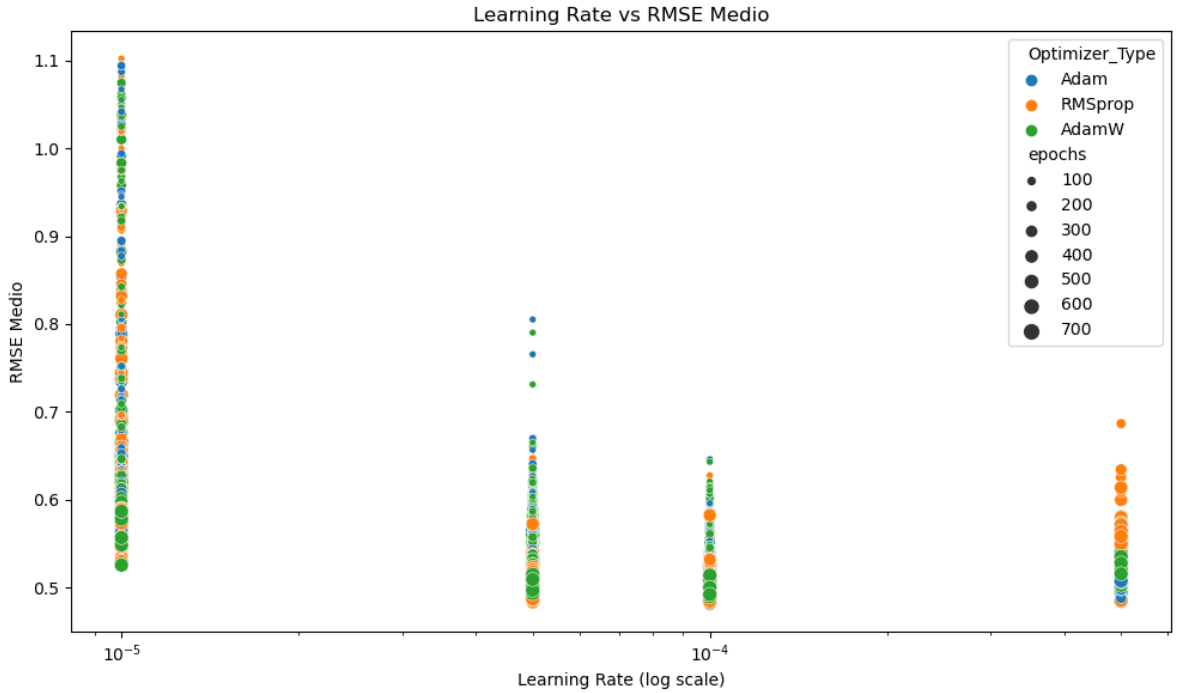


**Figura 4.5:** Confronto tra le learning rate e RMSE medio

con AdamW), suggerendo un equilibrio ottimale tra velocità di convergenza e stabilità dell'addestramento.

Il grafico dedicato (4.5) mostra l'andamento dell'RMSE medio in funzione del learning rate, stratificato per tipo di ottimizzatore, consentendo di osservare come la combinazione `AdamW` $+ 5 \times 10^{-4}$ risulti la più performante nel complesso.

### 4.2.6 Discussione complessiva

L'analisi complessiva della MLP mostra come il modello riesca a catturare le relazioni tra variabili in maniera soddisfacente, con buone performance soprattutto per le metriche relative agli ordini e al tasso di conversione, caratterizzate da bassa variabilità e RMSE contenuti. Tuttavia, la predizione di click e spesa rimane più complessa, come evidenziato dai valori medi di RMSE più elevati e da correlazioni forti tra le metriche di errore.

Dal punto di vista degli iperparametri, emerge chiaramente che la scelta della dimensione del batch e del learning rate ha un impatto molto più marcato rispetto alla selezione dell'ottimizzatore. In particolare, batch più piccoli e learning rate moderati consentono di ottenere i risultati migliori.

Queste osservazioni saranno fondamentali per orientare la successiva fase di integrazione della MLP nel framework di contextual bandit, dove la stabilità e l'accuratezza delle predizioni iniziali rivestono un ruolo centrale nel contenimento del regret cumulativo.

## 4.3 Confronto tra le Implementazioni del Contextual Bandit

Dopo aver analizzato nel dettaglio il funzionamento del Contextual Bandit e le sue varianti, questa sezione è dedicata al confronto tra i due approcci principali discussi nel capitolo 3.4. In particolare, si andrà a valutare come le modifiche introdotte nella seconda implementazione abbiano influenzato i risultati ottenuti, con particolare attenzione alla variazione dei valori di *regret* e alla coerenza scientifica dei dati.

Come ricordato, nella prima implementazione il calcolo della funzione di ricompensa era basato direttamente sui dati reali, generando valori anomali di *regret*, spesso negativi e di magnitudine elevata. Tale comportamento ha portato a risultati non plausibili, in quanto implicavano prestazioni superiori allo scenario reale di riferimento. La ridefinizione della funzione di ricompensa, basata invece sulle predizioni generate dal bandit stesso, ha consentito di riportare i risultati entro un intervallo coerente e interpretabile, con valori medi di *regret* compresi tra lo 0% e il 40%.

In questa sezione verranno dunque messi a confronto i due scenari, analizzando nel dettaglio gli effetti prodotti dal cambiamento nella definizione della ricompensa e valutando l'impatto delle diverse configurazioni testate (con e senza MLP, con MLP priva di pesi iniziali e con architetture più compatte). L'obiettivo è quello di mettere in evidenza in che misura la seconda implementazione abbia corretto le criticità iniziali e reso i risultati più affidabili, ponendo le basi per un utilizzo più robusto del Contextual Bandit nel contesto dell'ottimizzazione delle campagne pubblicitarie PPC su Amazon.

### 4.3.1 Analisi dei Risultati del Primo Modello

In questa sezione vengono presentati e commentati i risultati relativi al primo modello di Contextual Bandit sviluppato, includendo varianti con e senza rete neurale (MLP) e con o senza inizializzazione dei pesi. I dati sono accompagnati da rappresentazioni grafiche (boxplot, scatter plot, heatmap e lineplot) che permettono di interpretare in modo più chiaro le dinamiche osservate.

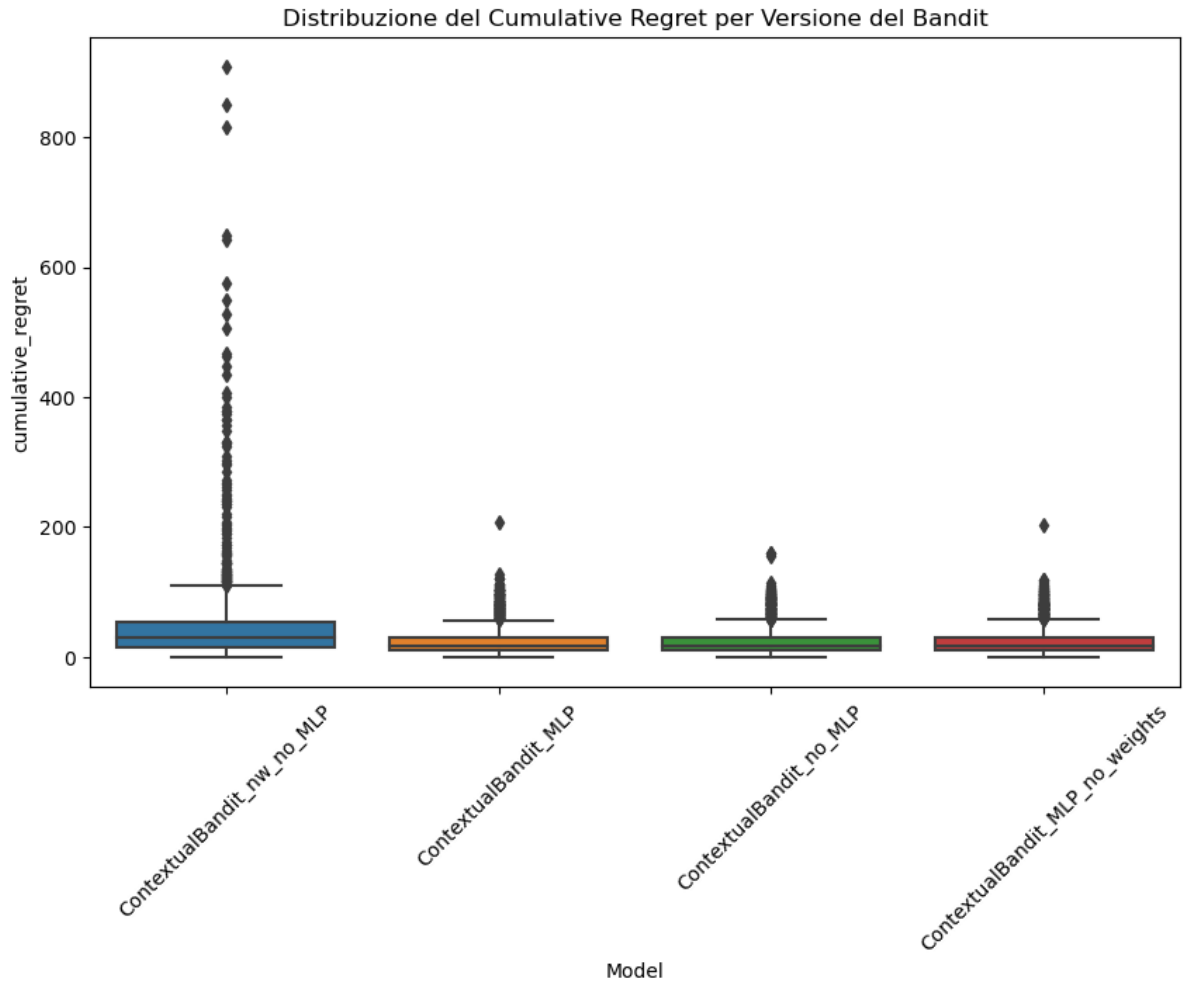


**Figura 4.6:** Distribuzioni del cumulative regret per ogni variante

### Statistiche descrittive del Cumulative Regret

L'analisi del *cumulative regret* evidenzia un comportamento differente tra le varianti di modello. Le versioni con `ContextualBandit_MLP` (da ora chiamata anche `MLP`) e `ContextualBandit_MLP_no_weights` (da ora chiamata anche "senza pesi) mostrano valori medi pressoché identici (circa 26.5), con una distribuzione abbastanza simile sia in termini di deviazione standard (24.5) che di valori estremi. La variante senza MLP (`ContextualBandit_nw_no_MLP`) si mantiene sugli stessi livelli medi (26.17), ma con un valore massimo più contenuto (160). La variante più instabile risulta essere la *ContextualBandit_nw_no_MLP*, che presenta un valore medio di 50 e una deviazione standard molto elevata (76), con picchi estremamente alti (max = 908). Questo indica una maggiore variabilità e una minore affidabilità.

### Relazione tra Cumulative Regret e Reward Medio in Test

Lo scatter plot mostra la relazione tra il *cumulative regret* e il reward medio. I dati confermano che modelli più instabili, come il *nw_no_MLP*, tendono ad avere valori di regret molto elevati, talvolta superiori a 400, a fronte di ricompense in media simili o solo leggermente inferiori rispetto agli altri modelli. Le varianti con MLP, invece, mantengono valori di regret contenuti con reward in linea, mostrando una maggiore coerenza.

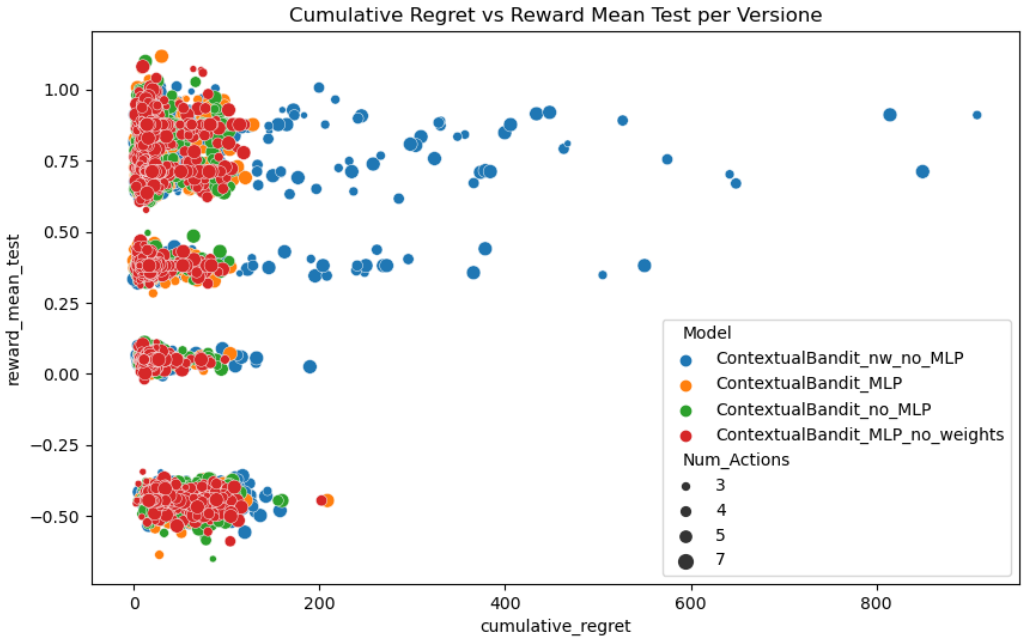


**Figura 4.7:** Scatter plot Distribuzioni del reward medio per ogni variante

### Sensibilità agli Iperparametri: Learning Rate e Batch Size

La heatmap evidenzia come i valori di regret crescano sensibilmente con l'aumento del learning rate. Per valori molto bassi di $lr$ ($10^{-5}$), i regret medi si attestano intorno a 18–20, mentre già a $lr = 10^{-3}$ si osserva un aumento netto (30–36). A $lr = 5 \cdot 10^{-3}$ i valori di regret crescono in modo considerevole, raggiungendo picchi superiori a 80. L'effetto del batch size appare meno marcato: batch size maggiori riducono leggermente il regret, ma l'impatto è secondario rispetto al tasso di apprendimento.

### Reward Medio in Test

Tutte le varianti di modello producono un reward medio pressoché identico (0.315), con deviazioni standard molto simili (circa 0.476). Questo suggerisce che, indipendentemente dall'architettura, il reward ottenuto in test rimane stabile. Le differenze tra i modelli emergono quindi principalmente sul fronte della gestione del regret, più che sulla massimizzazione del reward medio.

### Lost Regret Medio

Il confronto sul *lost regret* rivela valori molto divergenti. Il modello con MLP presenta un valore medio negativo marcato (-316) e un'elevata varianza, mentre il modello senza MLP mostra valori positivi (257). La variante senza pesi iniziali riduce l'entità delle anomalie (-142), mentre la versione *nw_no_MLP* si mantiene positiva (67). Questo dato conferma l'instabilità del calcolo della funzione di ricompensa nelle prime implementazioni e la difficoltà di ottenere valori coerenti.

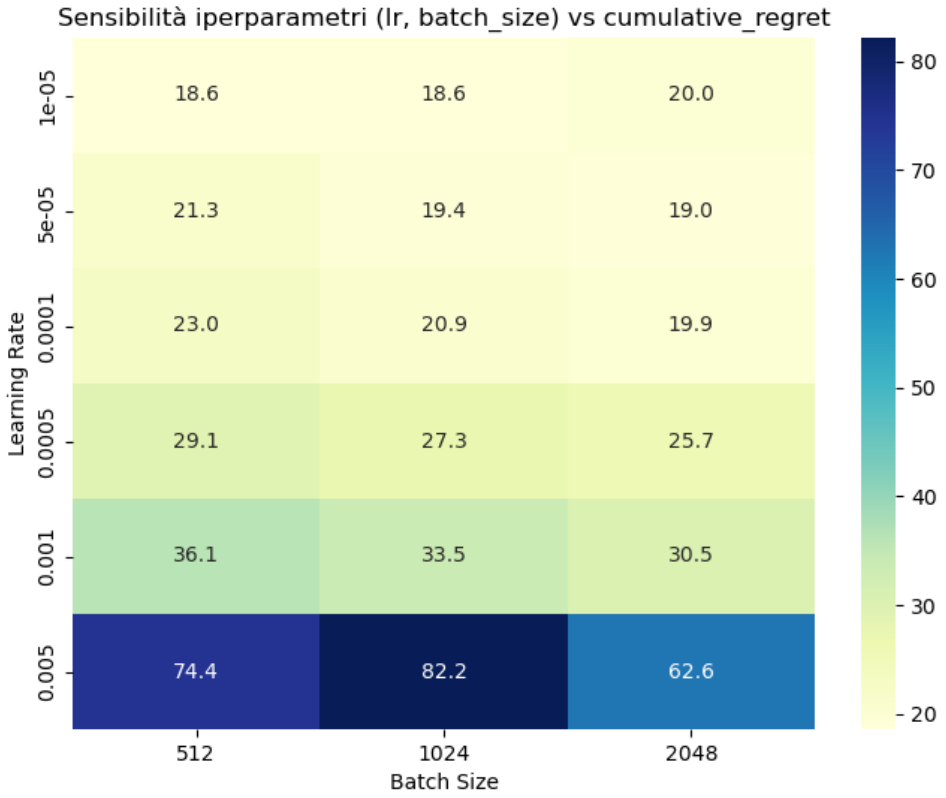


**Figura 4.8:** Heatmap di confronto tra Batch size, learning rate e cumulative regret

### Impatto del Learning Rate sul Cumulative Regret

L'analisi più dettagliata per modello e numero di azioni conferma quanto osservato nella heatmap: learning rate più bassi ($10^{-5}$) garantiscono regret medi più contenuti (14–22), mentre valori crescenti portano a un peggioramento progressivo. Il modello *nw_no_MLP* è il più sensibile, con incrementi drastici del regret già a $lr = 10^{-3}$ e valori estremamente elevati ($> 200$) a $lr = 5 \cdot 10^{-3}$.

### Impatto del parametro $k$ sul Cumulative Regret

L'effetto del parametro $k$ (Fig. 4.12), calcolato con la formula 3.2 presente nel capitolo 3.4.3, mostra una chiara tendenza: valori bassi (0.2–0.3) mantengono il regret medio su livelli contenuti (15–28), mentre valori più elevati (0.7–1.0) comportano un aumento sostanziale, in particolare per i modelli senza inizializzazione dei pesi e senza MLP. Ad esempio, con $k = 1.0$ e $action = 7$, il regret medio raggiunge valori oltre 76. Ciò evidenzia la necessità di una calibratura accurata di questo iperparametro per contenere la crescita del regret.

### Risultati Optimal input

In questa sezione vengono approfondite analisi tecnico-stastitiche sui risultati ottenuti dalla funzione riportata nel capitolo 3.4.4. Tale funzione interroga il modello contextual

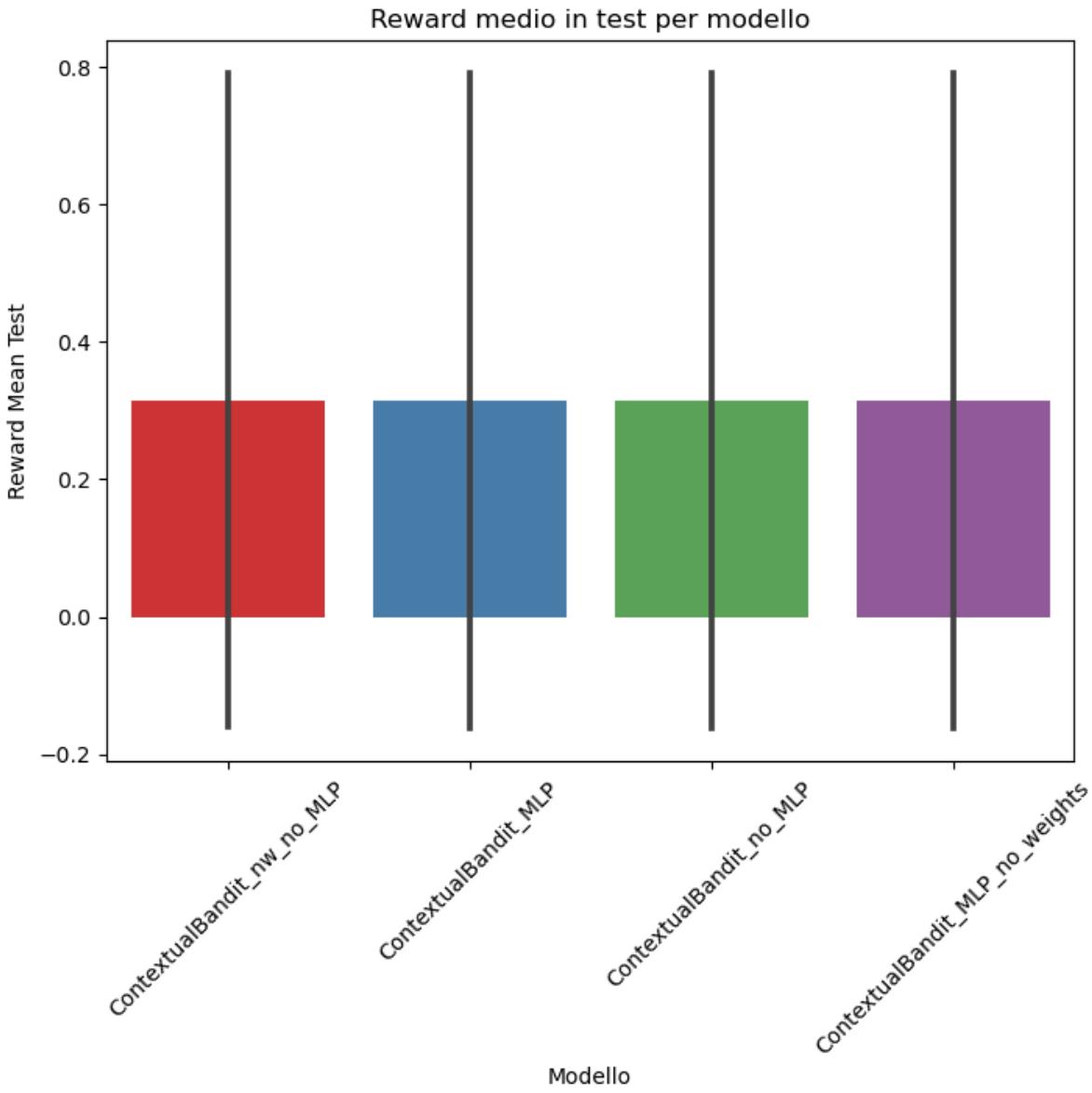


**Figura 4.9:** Boxplot del reward medio

bandit per ottenere le migliori configurazioni di click, keyword, spesa e altre feature al fine di fornire le dieci migliori strategie operative da applicare alle pubblicità.

### Optimal input: Correlazione tra le Feature

### Identificazione delle relazioni statistiche principali

I grafici forniscono insight sulla correlazione tra le variabili principali degli input e il reward previsto, utilizzando una heatmap delle correlazioni e un boxplot delle distribuzioni di reward.

**Correlazioni (Heatmap)** Nel primo grafico (Fig 4.14), la heatmap mostra la correlazione tra variabili chiave come "`lancio`", "`performance`", "`Day_of_Week`", "`Match Type Encoded`" e "`similarity`". I valori di correlazione più alti in termini assoluti sono:

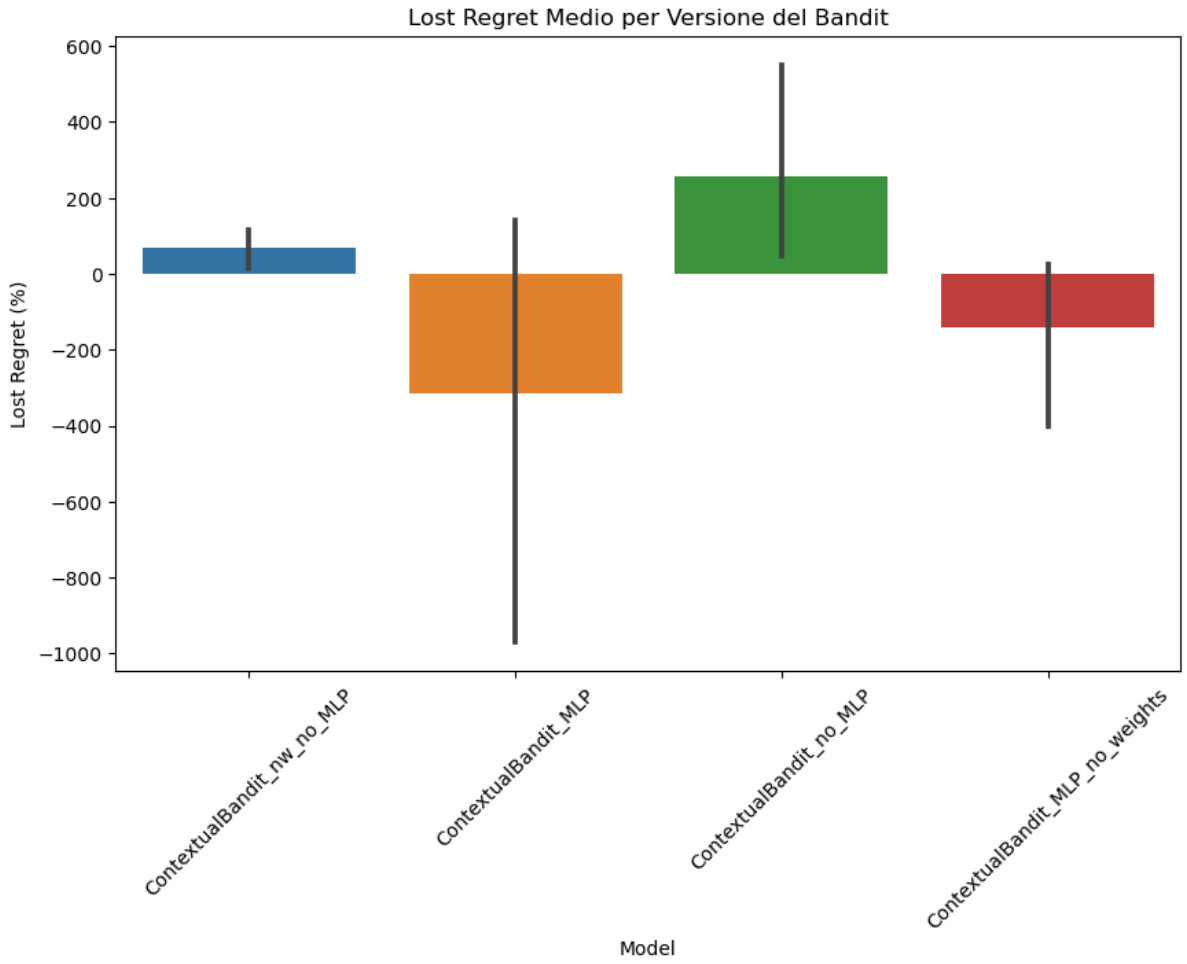


**Figura 4.10:** Boxplot del lost regret per versione

- Correlazione positiva ($r = 0.69$) tra "`Match Type Encoded`" e "`Day_of_Week`", indicando che determinati tipi di match si allineano con specifici giorni della settimana. Questo comportamento non sembra catturare la realtà dei fatti in quanto non sono due variabili dipendenti.
- Correlazione negativa ($r = -0.65$) tra "`performance`" e "`Match Type Encoded`", suggerendo che un tipo di match più preciso tende a indicare una campagna dove non è stata prevista una strategia forte, forse per la natura selettiva e restrittiva delle keyword.
- Correlazione negativa ($r = -0.41$) tra "`performance`" e "`Day_of_Week`", indicando che la `performance` può variare in base ai giorni della settimana.

### Osservazione delle distribuzioni di reward (Boxplot)

Il secondo grafico (fig. 4.13) presenta i boxplot delle distribuzioni di reward per diverse varianti di input. In particolare:

- La variante "optimal_inputs" che indica la variante con MLP di supporto mostra un reward medio più alto (circa 0.8) con una dispersione contenuta, suggerendo che è la configurazione migliore in termini di stabilità e performance.

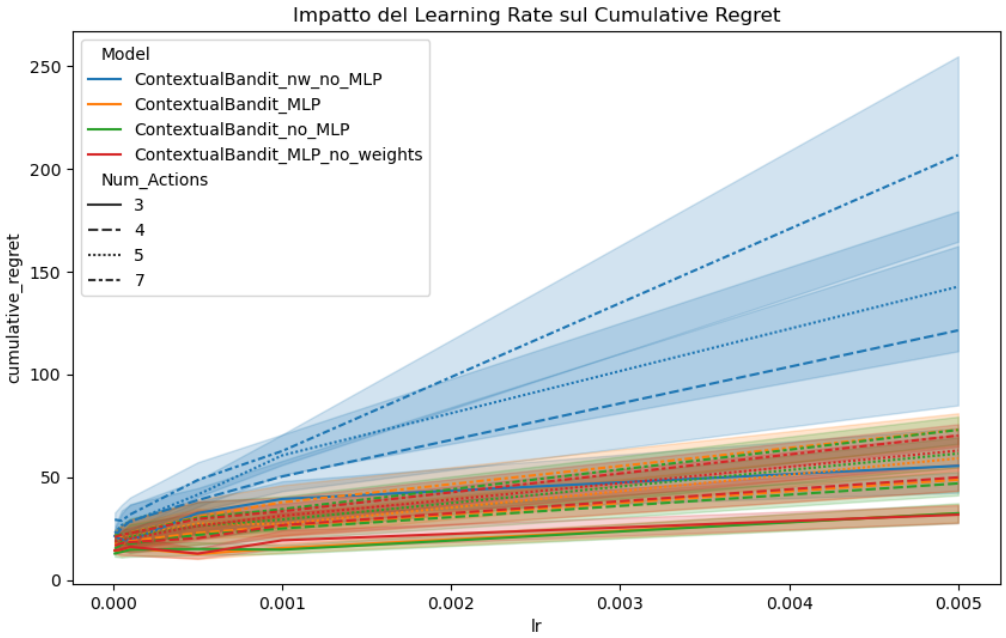


**Figura 4.11:** impatto del learning rate sul cumulative regret

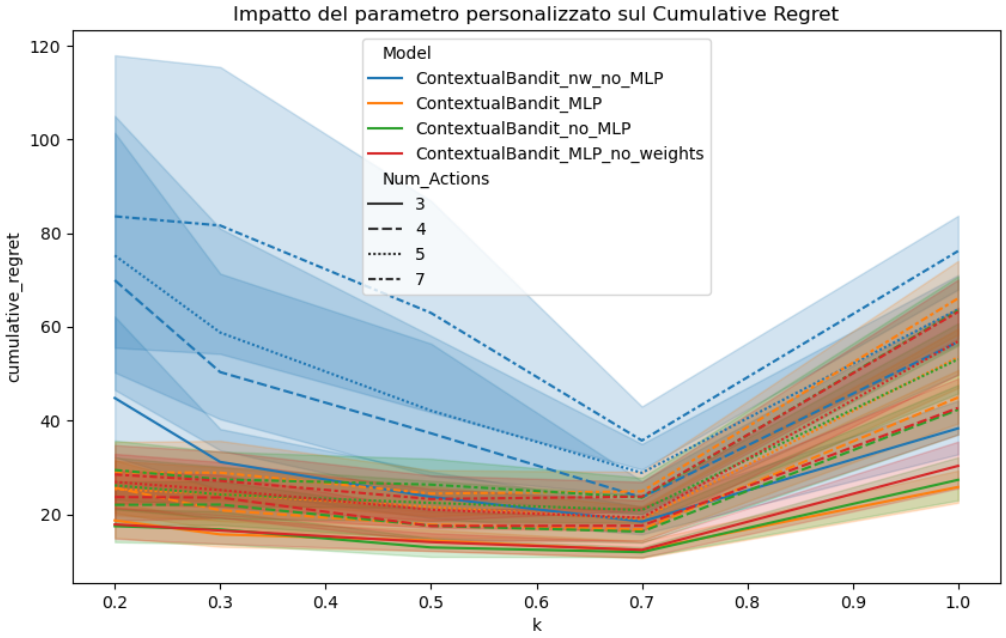


**Figura 4.12:** Variazione del cumulative regret in base al parametro K

- La variante "optimal_inputs_no_MLP" mostra un reward medio inferiore, con una maggiore dispersione, suggerendo che la mancanza della MLP non permette al modello di massimizzare correttamente il reward
- La variante "optimal_inputs_MLP_no_weights" ha una media più bassa rispetto alla configurazione MLP standard, ma con una deviazione standard contenuta,

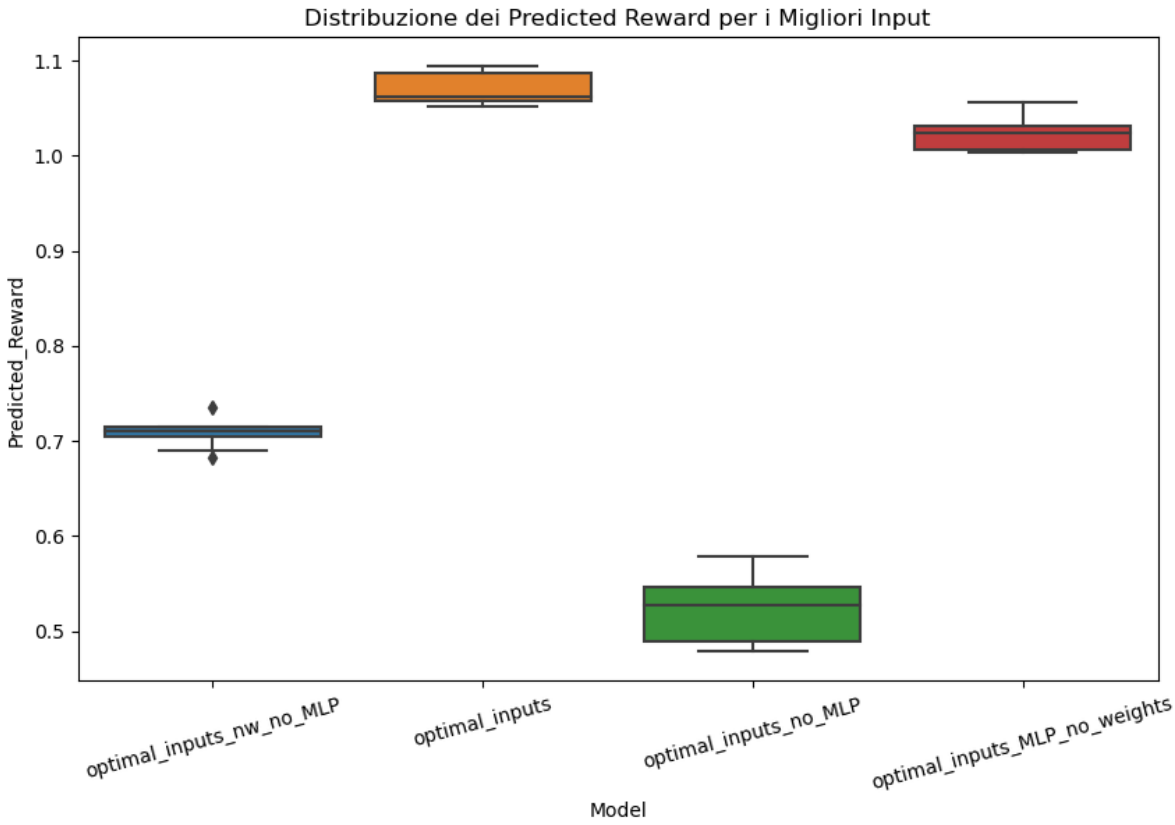


**Figura 4.13:** Distribuzioni del predicted reward per i migliori input

indicando che l'assenza di pesi nella rete MLP riduce la varianza ma non migliora la performance.

- La variante "optimal_inputs_nw_no_MLP" Ha una media inferiore sia al modello base sia al modello senza pesi nella MLP e ha la varianza più piccola tra tutte le varianti, indicando un modello stabile ma poco performante.

### Sintesi e implicazioni pratiche

In sintesi, il modello che utilizza l'MLP con i pesi sembra essere il più promettente, con un reward medio alto e una bassa deviazione standard. Questo suggerisce che le configurazioni che sfruttano una rete neurale MLP di supporto, soprattutto quando ottimizzate per input specifici, sono più robuste e performanti nel contesto delle campagne PPC su Amazon.

D'altra parte, le varianti che non utilizzano l'MLP o non utilizzano pesi specifici tendono ad avere performance inferiori o più variabili, indicando che il modello MLP gioca un ruolo cruciale nel migliorare la predizione dei reward.

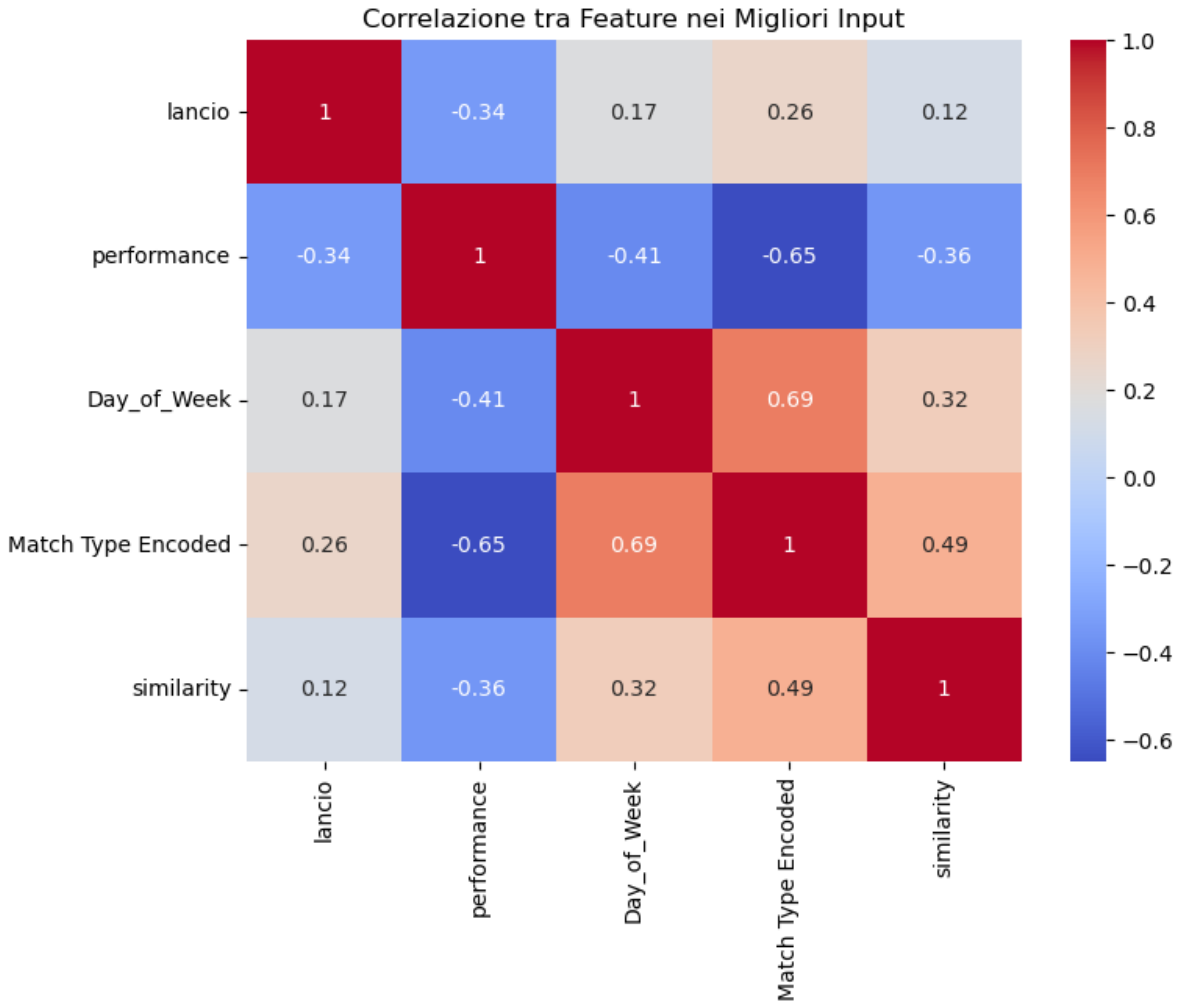


**Figura 4.14:** Heatmap per il cumulative regret per ogni variante

### 4.3.2 Analisi dei Risultati della Seconda Variante del Modello

In questa sezione vengono riportati e analizzati i risultati ottenuti dalla seconda variante del modello di Contextual Bandit, caratterizzata dall'introduzione della logica di selezione basata sul calcolo del reward su dati predetti. La lettura dei dati è accompagnata da rappresentazioni grafiche che supportano l'interpretazione dei fenomeni osservati.

#### Statistiche descrittive del Cumulative Regret

Le statistiche (Figura 4.15) mostrano che i modelli con MLP, con e senza inizializzazione dei pesi, e quello senza MLP presentano valori medi molto simili di *cumulative regret* (circa 35.5), con deviazioni standard elevate (69), indicando un'elevata variabilità. La variante *nw_no_MLP* si distingue con un valore medio inferiore (27.18) e una deviazione standard leggermente più contenuta (62.77). Si osserva anche la presenza di valori minimi negativi, che riflettono possibili situazioni di sovra-ottimizzazione o anomalie nei dati. La distribuzione delle quartili conferma che la mediana è più bassa (intorno a 10), ma le code lunghe causano valori massimi superiori a 200 in quasi tutte le varianti.

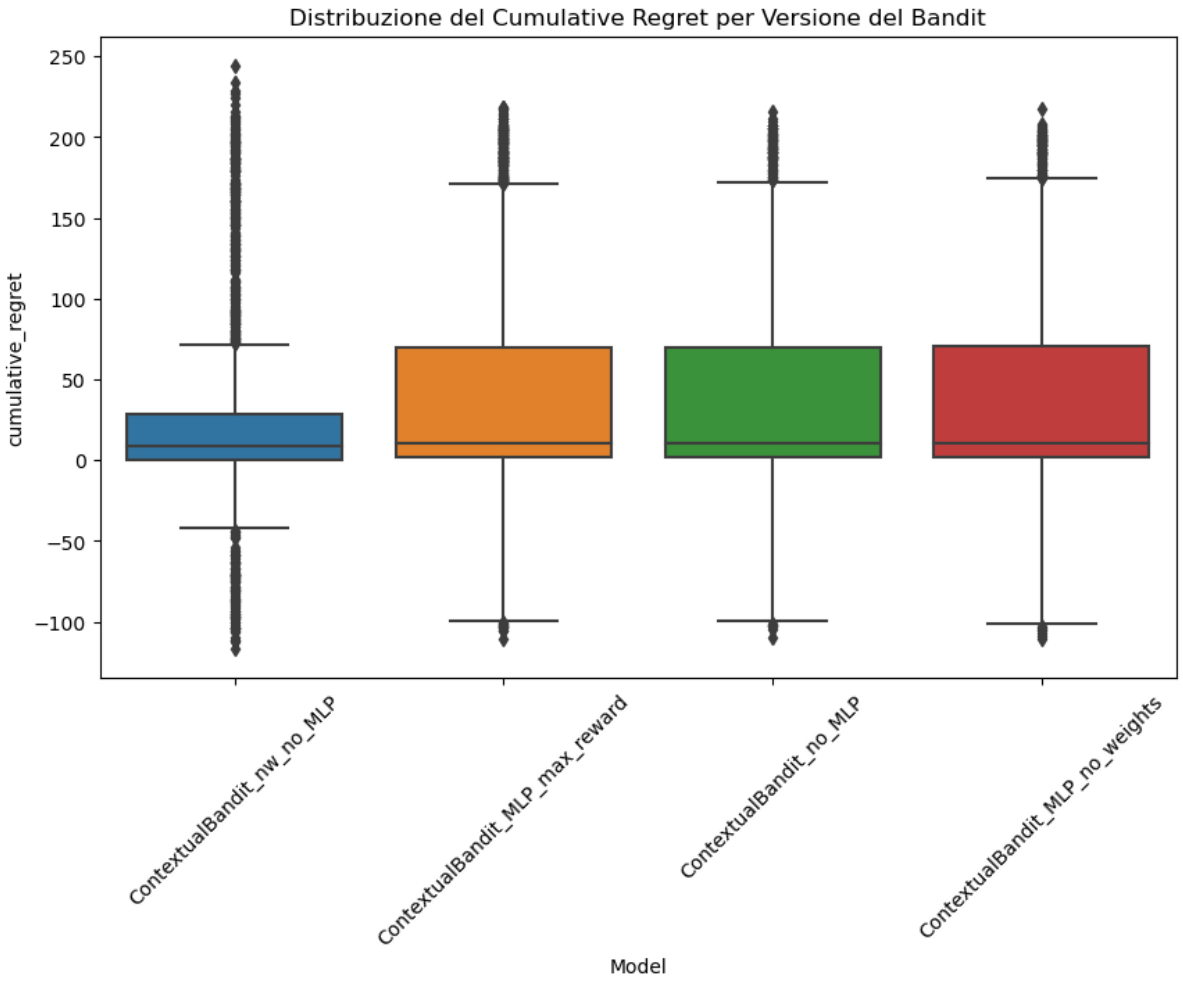


**Figura 4.15:** Distribuzioni del cumulative regret per ogni variante

### Relazione tra Cumulative Regret e Reward Medio in Test

Lo scatter plot (Figura 4.16) mostra un'interessante dinamica: mentre la maggior parte dei modelli mantiene valori di regret positivi associati a reward medi positivi, alcune configurazioni (in particolare le varianti con MLP senza pesi) evidenziano regret negativi accompagnati da valori di reward medio negativi (fino a -0.44). Questo indica instabilità nei processi di ottimizzazione, con possibili oscillazioni tra fasi di apprendimento profittevoli e fasi in cui il modello peggiora le proprie performance.

### Sensibilità agli Iperparametri (Learning Rate e Batch Size)

La heatmap (Fig 4.17) evidenzia un chiaro effetto del learning rate sul *cumulative regret*. Valori bassi ($10^{-5}$) portano a regret molto elevati (oltre 70), mentre incrementando progressivamente il learning rate si osserva una riduzione consistente, fino a valori minimi con $lr = 0.005$ (circa 4.6–5.5). Il batch size influenza in modo meno marcato i risultati: a parità di learning rate, batch più grandi tendono a mantenere valori leggermente più alti di regret, ma senza differenze sostanziali rispetto all'impatto del $lr$.

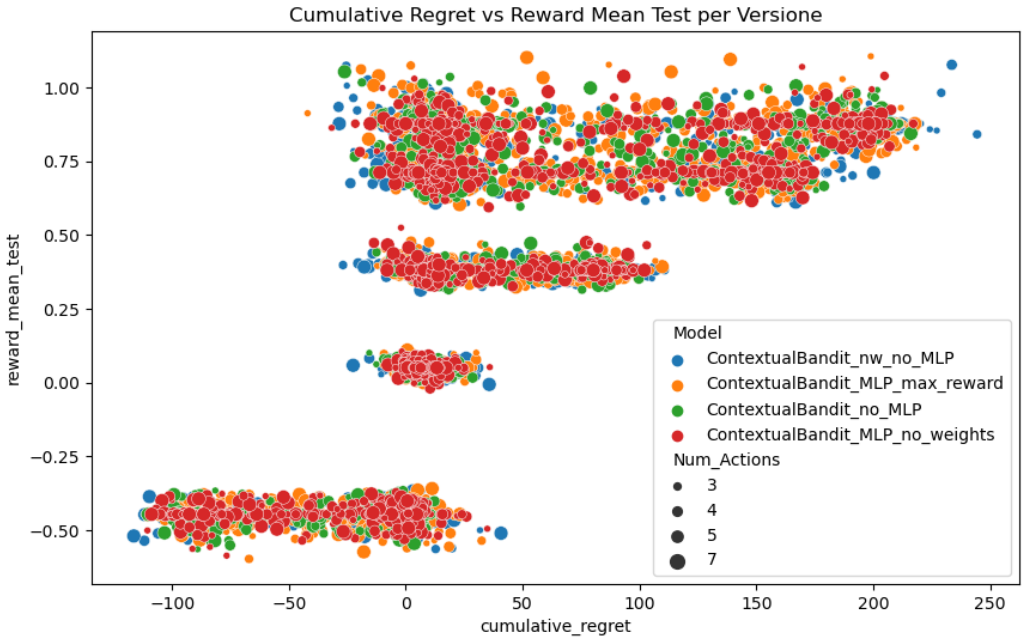


**Figura 4.16:** Scatter plot Distribuzioni del reward medio per ogni variante

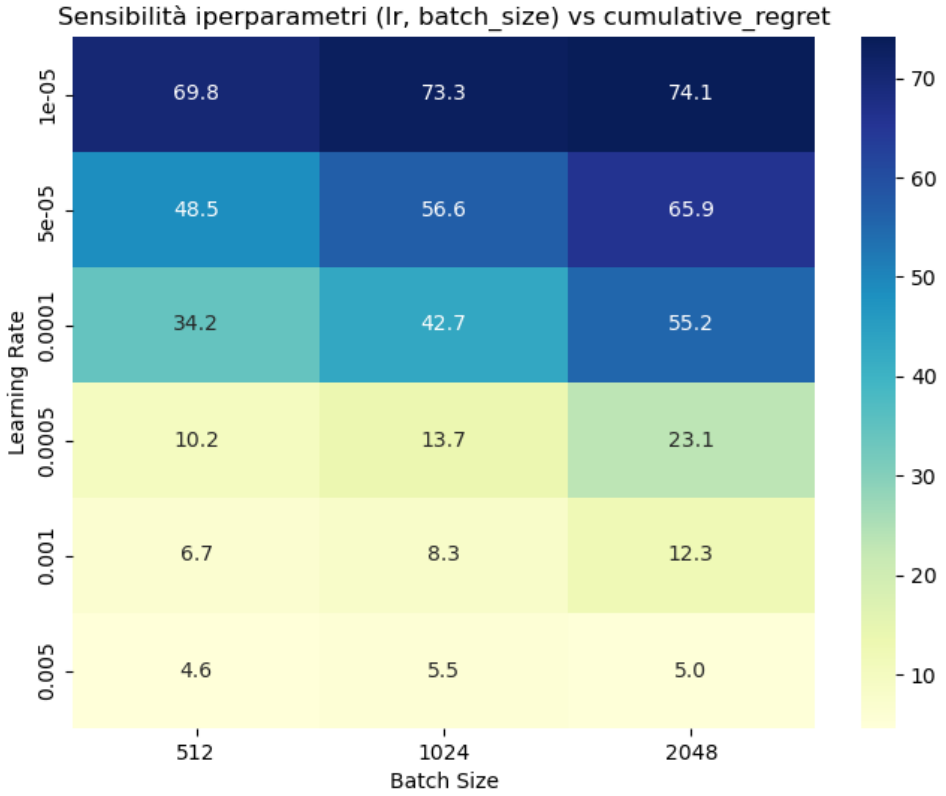


**Figura 4.17:** Heatmap di confronto tra Batch size, learning rate e cumulative regret

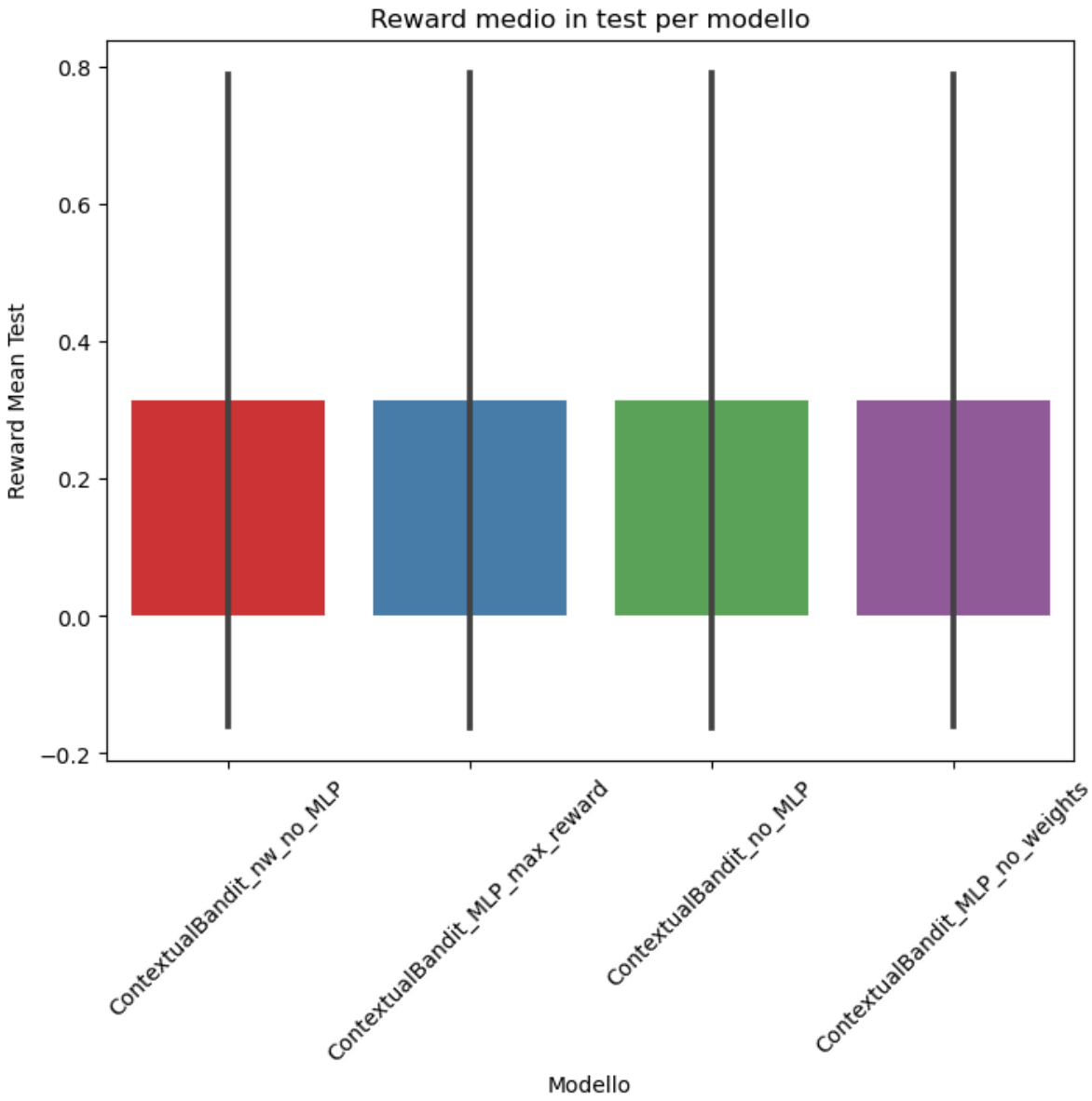


**Figura 4.18:** Boxplot del reward medio

### Reward Medio in Test

Il reward medio in test rimane pressoché invariato tra le varianti (0.314–0.315), con deviazioni standard intorno a 0.476. Questi risultati (Fig 4.18) confermano che, nonostante le differenze nei valori di regret, la capacità media di generare reward rimane costante tra i modelli.

### Lost Regret Medio

I valori di *lost regret* (Figura 4.19) risultano positivi per tutte le varianti, con medie tra 34 e 41. Ciò rappresenta un netto cambiamento rispetto alla prima variante, dove erano stati osservati valori negativi. Le deviazioni standard (intorno a 38–42) indicano una moderata variabilità, ma complessivamente i risultati appaiono più stabili e coerenti.

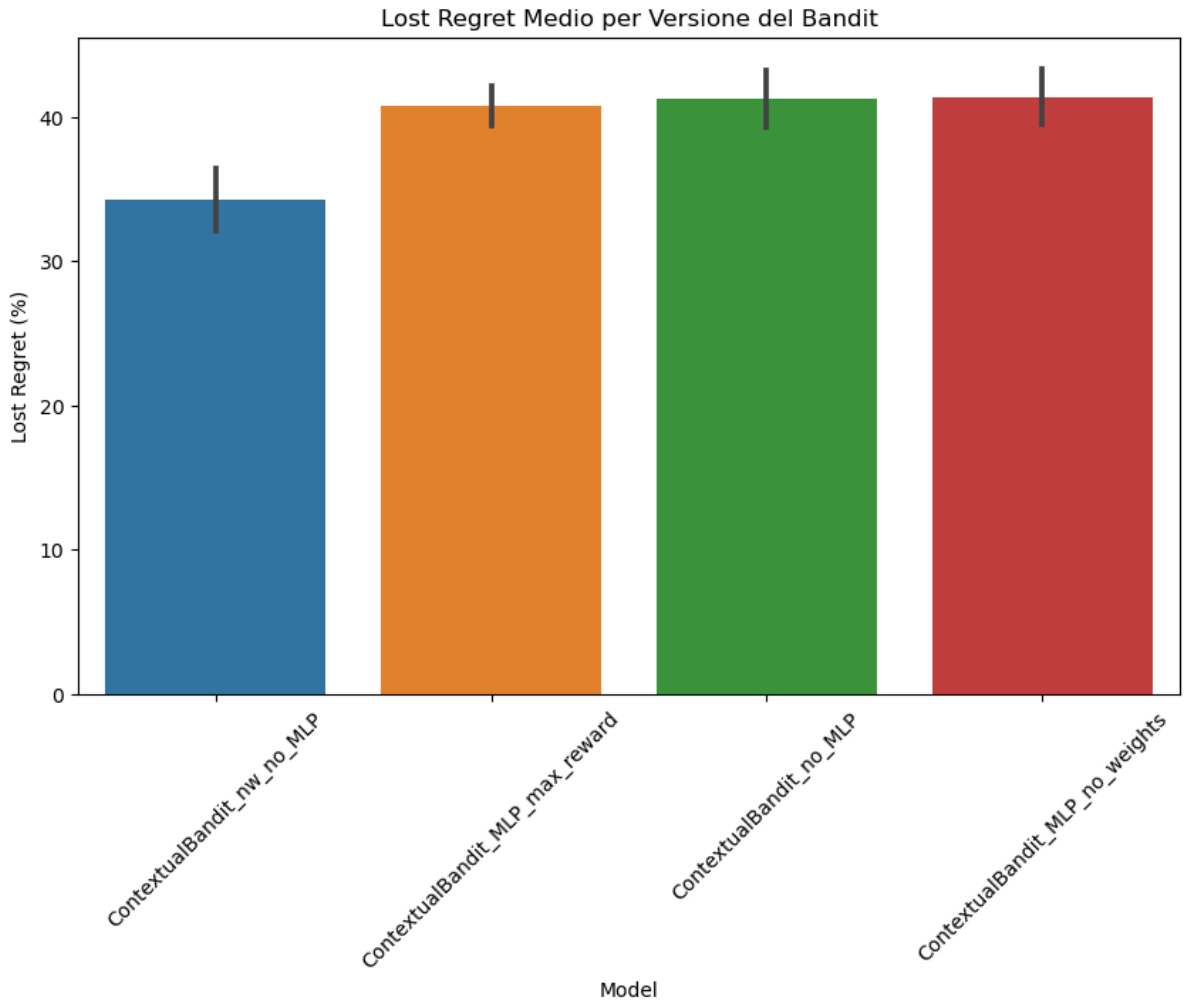


**Figura 4.19:** Boxplot del lost regret per versione

### Impatto del Learning Rate sul Cumulative Regret

Un'analisi più dettagliata (Fig. 4.20)conferma che valori bassi di learning rate ($10^{-5}$) conducono a regret medi molto elevati (oltre 70 per il modello con MLP), mentre aumentando $lr$ fino a $5 \cdot 10^{-3}$ si ottengono riduzioni significative, con valori medi inferiori a 6. Anche in questo caso, la variante *nw_no_MLP* mostra maggiore stabilità ai valori elevati di $lr$, mantenendo regret molto contenuti (tra 4.6 e 6.2).

### Impatto del parametro $k$ sul Cumulative Regret

L'analisi sul parametro $k$ (Fig. 4.21) evidenzia una forte sensibilità del modello MLP: a valori bassi di $k$ (0.2), i regret medi sono molto alti (circa 98), mentre con valori crescenti di $k$ il regret tende a ridursi progressivamente. Per la variante *nw_no_MLP*, al contrario, si osserva un comportamento più instabile: a $k = 1.0$ emergono addirittura valori medi negativi (fino a -29), segno di anomalie nella gestione delle scelte ottimali.

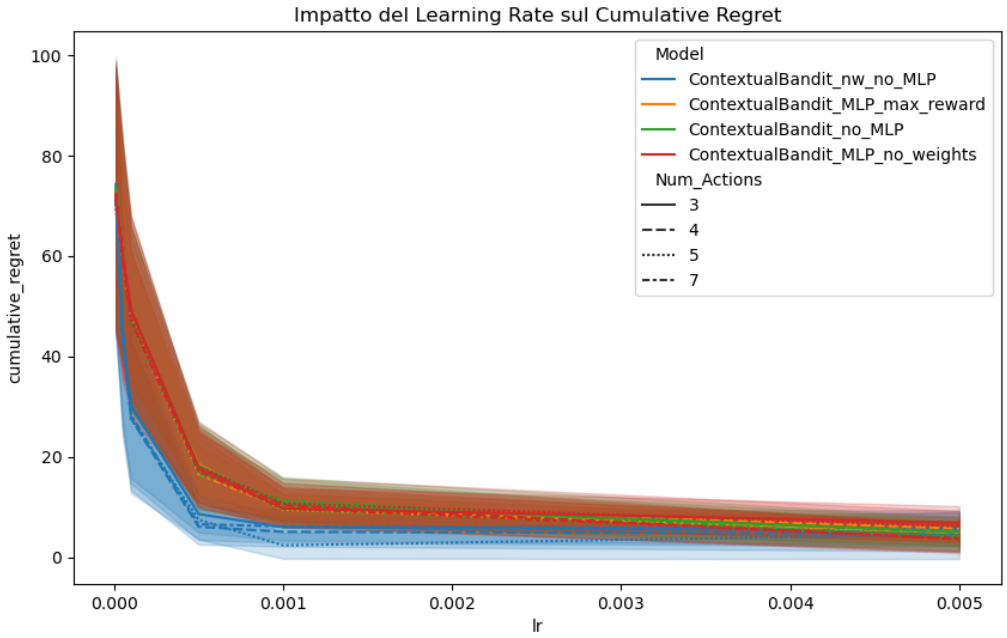


**Figura 4.20:** impatto del learning rate sul cumulative regret

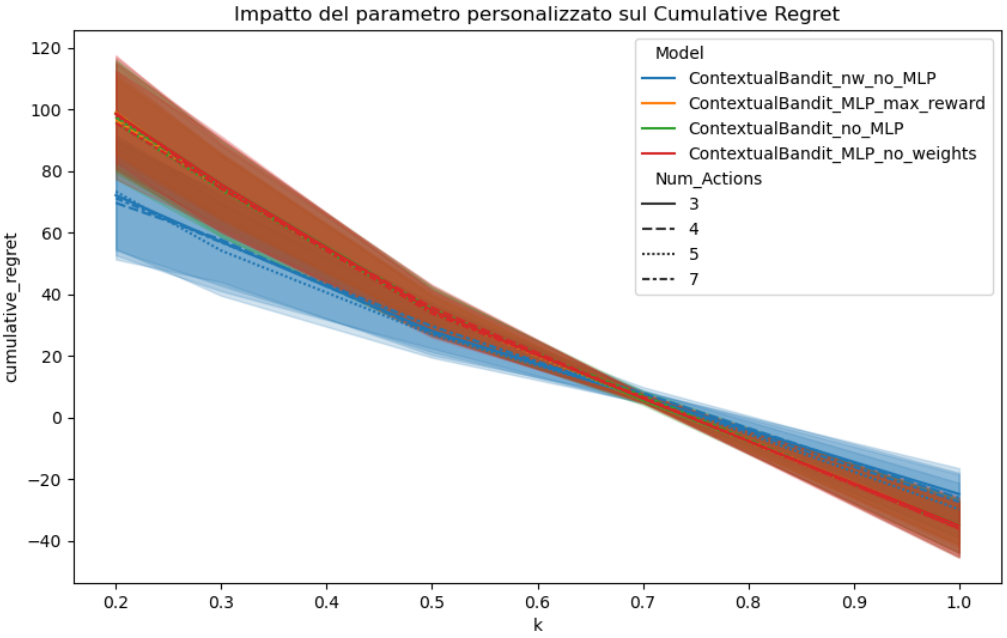


**Figura 4.21:** Variazione del cumulative regret in base al parametro K

### Commento al heatmap di correlazione ("Correlazione tra Feature nei Migliori Input")

Il grafico mostrato (fig. 4.23) rappresenta la matrice di correlazione fra le feature selezionate calcolata sui migliori input individuati dal procedimento di ottimizzazione. Di seguito viene fornita un'interpretazione delle relazioni più significative osservate:

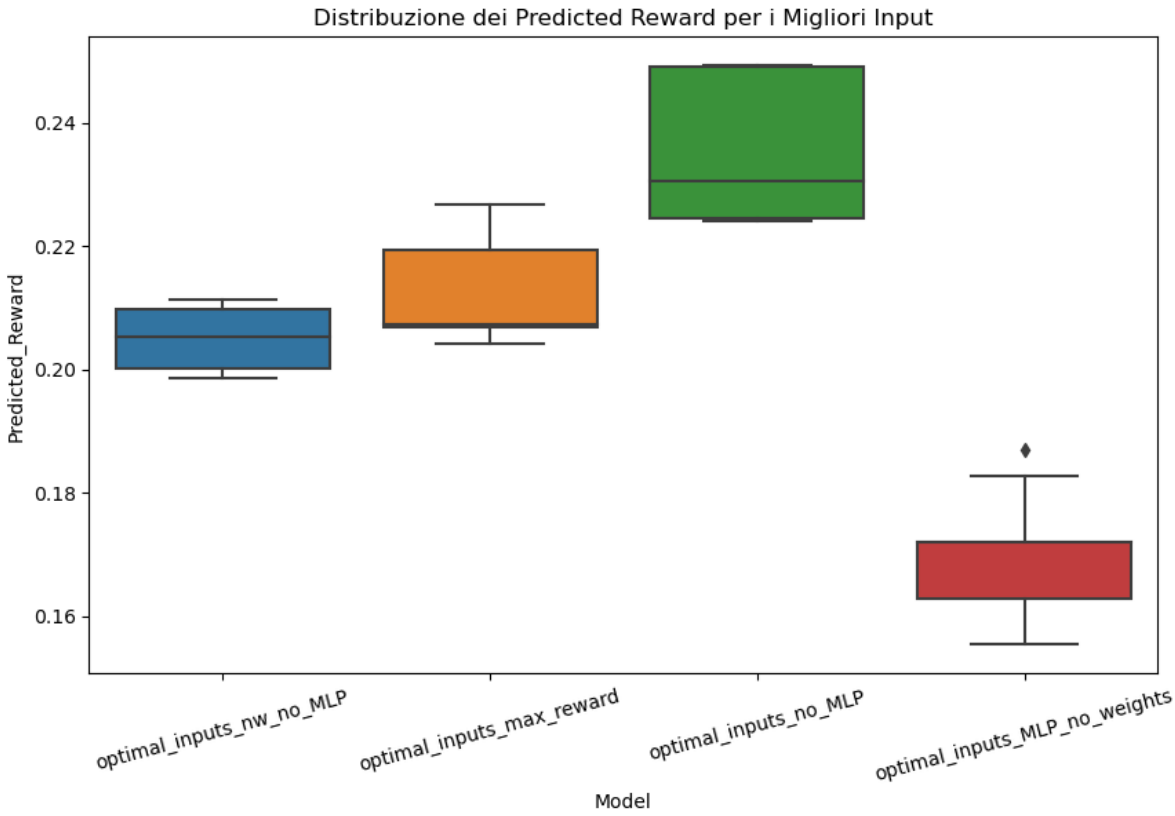


**Figura 4.22:** Distribuzioni del predicted reward per i migliori input

- **Correlazioni forti e interpretabili:**
  - *performance vs similarity* $\approx -0.72$: esiste una correlazione negativa marcata fra la variabile `performance` e la `similarity`. Ciò indica che, nei migliori input, contesti con elevata similarità semantica rispetto alla pagina prodotto tendono ad avere un valore di `performance` più basso. Un'interpretazione possibile è che le campagne etichettate come "performance" utilizzino keyword meno semanticamente vicine alla pagina prodotto (ad esempio keyword ad alto traffico ma meno specifiche). Possiamo quindi dire che il bandit cattura molto bene il meccanismo di come Amazon collega keyword specifiche (Match type: Exact) con la loro popolarità e flussi di ricerca.
  - *Match Type Encoded vs similarity* $\approx +0.69$: la correlazione positiva elevata suggerisce che tipologie di match più "stringenti" (es. exact/phrase, se codificate con valori maggiori) coincidono con search terms semanticamente più simili al prodotto. Questo è coerente con il comportamento atteso: corrispondenze più precise generano maggiore similarità semantica.
- **Correlazioni moderate e informazioni operative:**
  - *lancio vs Match Type Encoded* $\approx +0.30$ e *lancio vs similarity* $\approx +0.22$: le campagne di lancio sembrano predisposte verso match più precisi e keyword semanticamente rilevanti. In pratica, le campagne di lancio tendono a privilegiare keyword strette e con elevata similarità, probabilmente per massimizzare la pertinenza nelle fasi iniziali.

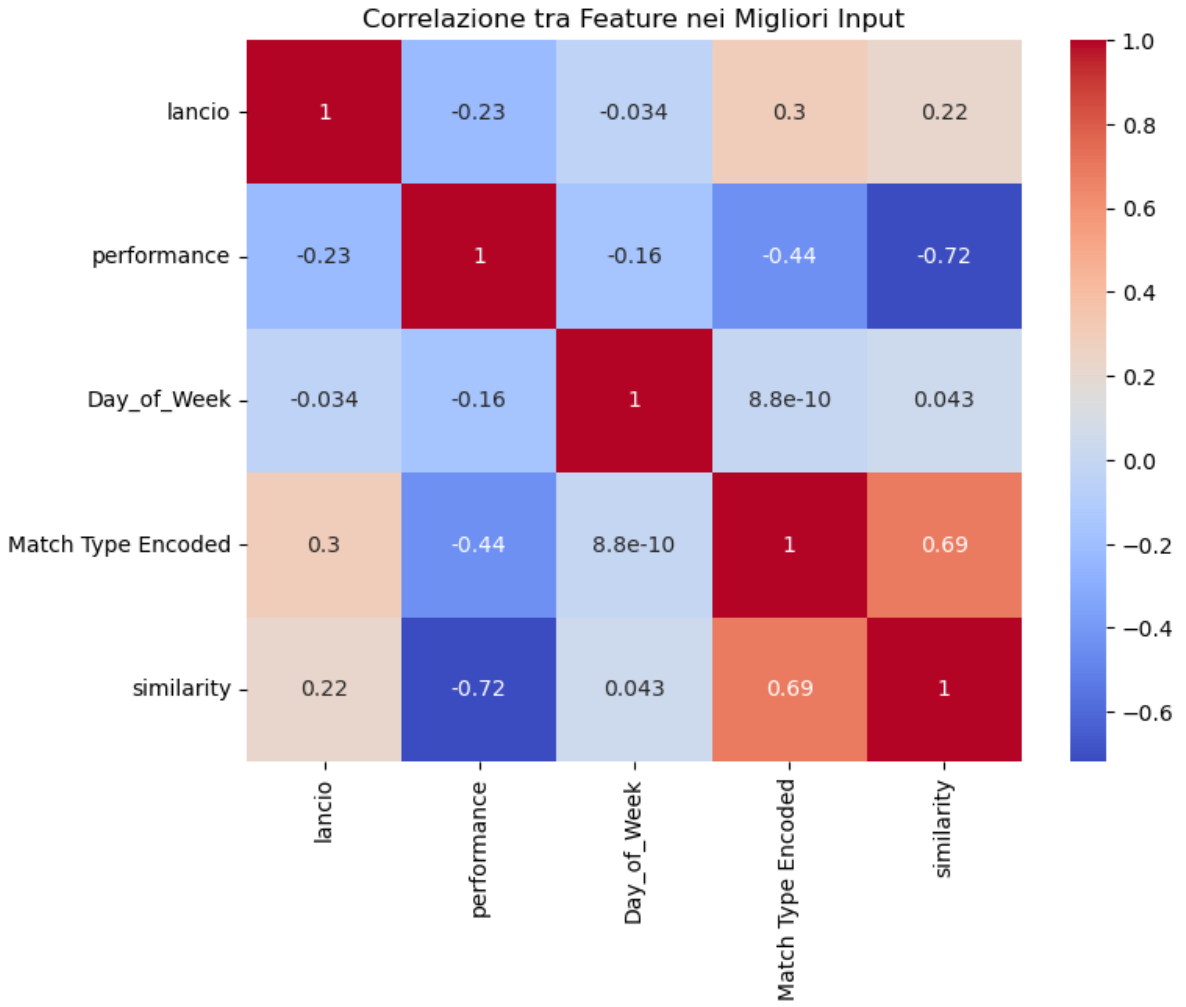


**Figura 4.23:** Heatmap per il cumulative regret per ogni variante

  - *lancio vs performance* $\approx -0.23$: c'è una moderata correlazione negativa fra il flag `lancio` e `performance`, che rafforza l'ipotesi che le campagne da lancio non coincidano necessariamente con quelle etichettate come "performance" (obiettivi e target diversi).

- **Feature quasi indipendenti:**

  - *Day_of_Week* risulta quasi disaccoppiata dalle altre variabili (valori molto prossimi a 0 con le altre feature). Questo suggerisce che, nel sottoinsieme dei migliori input, il giorno della settimana non è un driver discriminante nella selezione degli esempi con reward elevato.

**Conclusioni operative dalla heatmap:** la coppia (`Match Type`, `similarity`) emerge come fattore strutturante dei migliori input; inoltre la relazione negativa fra `performance` e `similarity` indica un trade-off strategico: keyword molto affini semanticamente al prodotto possono non appartenere alle campagne etichettate come "performance". Questo insight è utile per la selezione operativa delle keyword ottimali: non basta massimizzare la similarità — occorre bilanciare anche la strategia di campagna (`lancio` vs `performance`) in funzione dell'obiettivo commerciale.

**Commento al boxplot dei *Predicted Reward* per i migliori input**

Il secondo grafico (Fig. 4.22)mostra la distribuzione del *predicted reward* (valore stimato dal modello) per i top input raggruppati per variante sperimentale. Di seguito un'analisi precisa dei principali aspetti numerici e delle implicazioni pratiche.

**Sintesi numerica (valori principali dalle statistiche):**

- **optimal_inputs_max_reward** (variante "max_reward"): mean $\approx 0.2126$, std $\approx 0.0081$, 25% $\approx 0.2068$, 50% $\approx 0.2073$, 75% $\approx 0.2193$, max $\approx 0.2267$.
- **optimal_inputs_MLP_no_weights**: mean $\approx 0.1729$, std $\approx 0.0086$, median $\approx 0.1704$, range $\approx [0.1648, 0.1938]$.
- **optimal_inputs_no_MLP**: mean $\approx 0.1732$, std $\approx 0.0125$, median $\approx 0.1709$, range $\approx [0.1559, 0.1943]$.
- **optimal_inputs_nw_no_MLP**: mean $\approx 0.1108$, std $\approx 0.0020$, median $\approx 0.1098$, range $\approx [0.1092, 0.1150]$.

**Interpretazione e osservazioni:**

1. **Variante "max_reward" nettamente superiore in termini di reward previsto:** la distribuzione del gruppo `optimal_inputs_max_reward` si colloca chiaramente su valori più alti (mediana $\sim 0.207$, media $\sim 0.213$) rispetto alle altre varianti. Questo è coerente con l'obiettivo di quella variante, che esplicita una funzione di selezione orientata alla massimizzazione del reward stimato: la procedura tende a proporre input con reward atteso maggiore.
2. **MLP senza pesi vs no-MLP: comportamenti simili ma con diversa dispersione:** sia `MLP_no_weights` che `no_MLP` mostrano mediana intorno a 0.17; la variante senza MLP presenta però una varianza leggermente più alta (std maggiore), segno che la stabilità delle top candidates è leggermente inferiore quando non si usa la MLP (o quando la MLP non è inizializzata con pesi pre-allenati).
3. **Variante `nw_no_MLP` povera ma consistente:** il gruppo `nw_no_MLP` ha il valore medio più basso ($\sim 0.11$) ma anche la deviazione standard più piccola. Questo indica che, pur producendo top input con reward basso, lo fa in modo consistente: la procedura non esplora combinazioni ad alto reward in questa configurazione.
4. **Presenza di outlier e loro significato:** nelle distribuzioni di `MLP_no_weights` e `no_MLP` si osservano alcuni outlier (valori superiori o inferiori al box). Gli outlier superiori possono indicare casi particolarmente promettenti che però andrebbero verificati con le metriche di business (vendite, spesa, click). Gli outlier inferiori, in particolare per `nw_no_MLP`, suggeriscono che alcune configurazioni sono poco efficaci e probabilmente vanno scartate.

**Implicazioni per la scelta operativa dei top input:**

- Se l'obiettivo prioritario è massimizzare il reward stimato (metrica interna del bandit), la variante `max_reward` fornisce in media i migliori candidati e dovrebbe essere preferita per generare proposte operative da testare in ambiente reale.

- Tuttavia, la decisione finale non può basarsi esclusivamente sul valore di reward: occorre incrociare questi risultati con le predizioni delle metriche di business (vendite, spesa e click). Ad esempio, un input con reward elevato ma con `predicted_spend` molto alto o con anomalie nei click (es. valori estremi) potrebbe non essere ottimale dal punto di vista del margine.
- La variante `nw_no_MLP`, per quanto consistente, produce reward troppo bassi per essere considerata competitiva; può comunque essere utile come baseline stabile per confronto.

**Nota su coerenza fra heatmap e distribuzioni di reward:** l'elevata correlazione positiva tra `Match Type` e `similarity` (0.69) suggerisce che le top candidates della variante `max_reward` probabilmente includono match più stringenti e keyword semanticalmente vicine al prodotto questo spiega in parte l'aumento del reward medio osservato per tale variante. Contemporaneamente, la correlazione negativa tra `performance` e `similarity` indica che molte di queste top candidates potrebbero non appartenere a campagne etichettate come "performance", rinforzando la necessità di valutare il trade-off strategico prima di applicare le modifiche in produzione.

**Alcuni spunti finali su questi risultati:** usare la variante `max_reward` per generare un primo set di candidate, ma validare ciascuna proposta con le predizioni di vendite/-spesa/click e con regole di sanity-check (clipping o soglie su click/impression) per evitare di promuovere input che, pur avendo reward stimato alto, comportino rischi operativi o anomalie nelle metriche di business. Questo tema verrà trattado in maniera più approfondita nel successivo capitolo.

# Capitolo 5

# Conclusioni

In questo elaborato si è inteso sperimentare e valutare alcune tecniche di *machine learning* come strumento di supporto alle aziende, con l'obiettivo di renderle più competitive in un mercato caratterizzato da forte dinamicità e da un'elevata intensità concorrenziale. L'impiego di approcci avanzati, quali quelli approfonditi nei capitoli precedenti, si rivela infatti fondamentale per affrontare la complessità dei dati disponibili e per sfruttare al meglio le potenzialità insite nelle informazioni che, diversamente, resterebbero parzialmente inaccessibili o difficilmente interpretabili dagli operatori. In particolare, nel dominio delle campagne pubblicitarie *Pay-Per-Click* (PPC) su piattaforme come Amazon, le dinamiche algoritmiche sottostanti risultano opache e non direttamente controllabili dall'utente, rendendo essenziale l'adozione di strumenti analitici e predittivi capaci di ottimizzare le decisioni strategiche.

Il presente capitolo, dedicato alle conclusioni, si propone di riprendere gli obiettivi inizialmente definiti all'avvio della ricerca e di valutare in che misura tali obiettivi siano stati raggiunti. Verranno quindi discussi i risultati conseguiti rispetto alle ipotesi di partenza, mettendone in evidenza sia gli aspetti positivi sia i limiti emersi nel corso della sperimentazione. In particolare, si analizzeranno criticamente gli esperimenti condotti, valutando l'efficacia dei modelli presentati e discutendo i fattori che hanno influito sulle loro prestazioni.

Un'attenzione specifica sarà rivolta anche al contributo che le metodologie adottate, come l'impiego di reti neurali unite agli approcci basati su contextual bandit, possono apportare alla letteratura scientifica di riferimento, ampliando le prospettive già esistenti e aprendo a nuove possibilità di ricerca. Infine, verranno discusse le possibili evoluzioni future di questo lavoro, evidenziando le direzioni di sviluppo più promettenti sia sul piano accademico sia su quello applicativo.

L'obiettivo ultimo di questo capitolo non è soltanto fornire una sintesi dei risultati ottenuti, ma anche proporre una riflessione complessiva sul valore del lavoro svolto e sulle sue potenzialità per la ricerca e per le applicazioni reali in contesti di mercato complessi e altamente competitivi.

## 5.1 Richiamo agli obiettivi della ricerca

Questo elaborato, come già discusso nella sezione 1.3, vuole analizzare le problematiche legate all'ottimizzazione delle campagne pubblicitarie *Pay-per-click* che basano il loro funzionamento sul concetto di aste o *bid*. La sfida che vuole affrontare questo elaborato è quella di rendere più vantaggioso per l'azienda l'equilibrio costi-ricavi automatizzando

la scelta delle keyword e il loro bid rispetto alle reali proposte di Amazon. In questo elaborato si è cercato di raggiungere attraverso l'approccio quantitativo, analizzando i dati e sviluppando i modelli che permetto in autonomia di massimizzare i ricavi minimizzando i costi.

L'elaborato affronta il tema della manipolazione e dell'analisi dei dati con l'obiettivo di estrarre la massima quantità di informazione utile da essi e di applicarla in un contesto ad alto valore strategico, quale l'ottimizzazione delle campagne pubblicitarie *Pay-Per-Click* (PPC) su Amazon. La fonte dei dati utilizzati è stata la piattaforma di gestione delle *Ads* di Amazon, dalla quale sono stati acquisiti report periodici contenenti informazioni relative alle performance delle campagne. Tali dati, successivamente, sono stati sottoposti a processi di *data cleaning* e trasformazione, con lo scopo di garantire la qualità del dataset ed estrarre variabili rilevanti per l'analisi.

Un aspetto centrale ha riguardato il trattamento delle *keyword*, le quali sono state convertite in vettori densi a 384 dimensioni mediante tecniche di rappresentazione semantica, al fine di aumentarne la capacità esplicativa e di consentire una più efficace modellazione dei rapporti tra query e prodotti. Tecniche sperimentali, descritte nella sezione 2.2.6, hanno inoltre permesso di stabilire un collegamento semantico tra ciascuna parola chiave e il prodotto di riferimento, rendendo possibile una rappresentazione più ricca e coerente del contesto pubblicitario. A valle di questa fase, è stata condotta un'analisi esplorativa del dataset per valutarne la struttura, la qualità e la quantità di informazione effettivamente estraibile, fornendo le basi per gli esperimenti successivi.

Come descritto nel Capitolo 3.2, sono stati dapprima implementati e testati diversi modelli di regressione tradizionali, tra cui LightGBM, XGBoost e Random Forest. In parallelo, si è posta attenzione sull'impiego di modelli di *machine learning* neurali, con particolare riferimento a differenti configurazioni di reti *Multi-Layer Perceptron* (MLP) (3.3.1). Tra le diverse varianti, è stato individuato il modello più performante e ne sono stati salvati i pesi per utilizzi successivi.

Il modello definitivo scelto per massimizzare i profitti e minimizzare i costi non si è tuttavia limitato a un approccio predittivo classico, bensì ha fatto ricorso a un algoritmo innovativo di tipo *Contextual Bandit*, approfondito nel Capitolo 3.4. Su tale algoritmo sono stati condotti ulteriori esperimenti, mirati a individuare le migliori combinazioni di parametri e strategie in grado di ottimizzare le scelte di spesa e allocazione delle keyword. I risultati complessivi degli esperimenti sono stati analizzati nel Capitolo 4, dove è stato inizialmente discusso il confronto tra i modelli di regressione classici e le MLP, concludendo che queste ultime hanno dimostrato prestazioni superiori. Successivamente, sempre nel Capitolo 4.3, è stata approfondita l'analisi delle performance del modello *Contextual Bandit*, evidenziando il suo contributo nell'ottimizzazione delle campagne PPC e nel miglioramento del bilancio tra costi sostenuti e profitti generati.

## 5.2 Sintesi dei risultati principali

In questa sezione verranno riassunti in breve quelli che sono stati i risultati dei modelli allenati, nello specifico verranno discussi solo i risultati dei modelli utilizzati per raggiungere l'obiettivo.

### 5.2.1 Valutazione complessiva dei risultati del MLP

L'allenamento delle reti neurali si è mostrato stabile nella previsione dei valori delle feature. Come già discusso nella sezione 4.2, i modelli addestrati riescono a catturare le complessità del dataset in maniera soddisfacente, soprattutto per metriche con magnitudo più piccola. La previsione invece di variabili come il click e la spesa rimane più complessa come viene mostrato dalle correlazioni tra le metriche di errore riportate in fig. 4.2, questa caratteristica può essere imputata allo scarso bilanciamento delle feature mostrato anche dai grafici in fig. 2.6. L'impatto dei parametri della rete come il Learning rate non si sono dimostrati particolarmente impattanti sull'allenamento del modello, ma è possibile affermare che per learning rate moderati e batch più piccoli è possbile ottenere il miglior modello. In generale le prestazioni della rete neurale non sono sufficienti a predire i valori target in maniera precisa, questo problema può essere imputato a i problemi seguenti

- **Problemi relativi al dataset**: Il dataset preso in esempio e spiegato nelle tabelle 2.1 e 2.2 contengono complessivamente più di 300.000 righe che riassumono le attività di circa 8 mesi di pubblicità su Amazon effettuata da Wellbeauty. Questo dataset presenta alcune problematiche:
  - *Scarsa rappresentatività delle strategie*: Seppur il dataset è stato acquisito attraverso la sezione di reportistica di Amazon e quindi è completo delle feature necessarie alla campagna e comprensivo dei risultati che ha ottenuto nei precedenti mesi, non è completo delle strategie alla quale una campagna può essere sottoposta, per esempio stategie di business rilegate a determinate campagne[1]. Si è cercato di interpretare attraverso i dati dei report e i nomi delle campagne a quale strategia facessero riferimento (come visto nella sez. 2.2.3) ma questa soluzione è troppo semplificativa e difficilmente generalizzabile dal modello.
  - *mancanza di keyword per le campagne automatiche*: come approfondito nella sezione 2.2.5, quando si ricorre all'uso di campagne automatiche, la keyword utilizzata dall'algorimo per promuovere il prodotto ad un cliente non è visibile, viene rimito il campo keyword con la stringa "*". Questa problematica è stata, dove possibile, risolta nella sezione 2.2.6. Nonostante i risultati più che soddisfacenti che sono stati ottenuti, molti dati vengono esclusi dall'allenamento in quanto non ci sono sufficienti informazioni a derivare ne ASIN del prodotto collegato ne la keyword originaria della ricerca.
  - *scarsità di dati* Nonostante la quantità di righe iniziali può essere considerata una buona quantità di dati, la fase di preprocessing ne riduce la quantità a meno di 30.000 righe. Questa forte diminuzione di campioni sulla quale addestrare il modello si dimostra insufficiente per una buona generalizzazione.
- **Problemi relativi al modello**: durante la fase di creazione delle reti neurali sono stati condotti degli esperimenti per provare diverse combinazioni delle reti come

[1]Le campagne di business possono essere create ad-hoc applicando strategie specifiche, come prezzi di bid elevati per pubblicizzare maggiormente il prodotto e aumentare la visibilità del brand, nonostante il prodotto venda in netta perdita molte aziende scelgono di sponsorizzare il loro brand tramite la piattaforma Amazon. Altre strategie messe in atto dalle aziende (e più volte messa in atto da Wellbeauty durante il periodo osservato) sono quelle che permetto al prodotto sponsorizzato di apparire in varie parti della pagina prodotto, come prodotto complementare o prodotto simile.

già visto nelle sezioni 3.3.1,3.3.1,3.3.1. Infine è stato scelto il primo modello per le sue prestazioni superiori agli altri. La struttura di questi modelli è stata creata manualmente ed è una struttura semplice e con pochi livelli. nonostante sia stata la struttura più semplice (3.3.1) ad aver ottenuto le migliori performance, non è da escludere che aumentando la complessità della sotto struttura della rete possano aumentare anche le performance. In rete non sono state trovate reti neurali con scopi simili a quelli presentati in questo documento di tesi ma si sono trovati due interessanti applicazioni di modelli che si avvicinano a questo scopo. Di seguito vengono brevemente spiegati:

- **shinleylee/Arbitrary_Distribution_Modeling (GitHub)** : Implementa il modello ADM (*Arbitrary Distribution Modeling, KDD'22*) basato su un Multi-Layer Perceptron (MLP) per prevedere la distribuzione dei prezzi di bid. Il repository contiene il codice Python e notebook di esempio (iPinYou, YOYI) per addestrare il modello ADM con la Neighborhood Likelihood Loss, ma non fornisce ne i pesi pre-addestrati pronti all'uso ne i dataset per emulare l'esperimento effettuato.

In sintesi, i risultati ottenuti confermano come l'impiego di una rete neurale MLP rappresenti un passo significativo verso una modellazione più flessibile e capace di catturare le complessità insite nei dati delle campagne pubblicitarie PPC. Tuttavia, le limitazioni discusse – legate sia alla natura del dataset sia alla struttura del modello – evidenziano come le prestazioni raggiunte non siano ancora sufficienti per garantire una predizione accurata e generalizzabile in contesti reali.

Un aspetto particolarmente rilevante riguarda la qualità e la completezza dei dati: la mancanza di keyword nelle campagne automatiche, la scarsità di informazioni riconducibili alle strategie aziendali adottate e la drastica riduzione dei campioni effettivamente utilizzabili per l'addestramento rappresentano ostacoli critici al pieno sfruttamento del potenziale delle reti neurali. Sul piano modellistico, sebbene le architetture semplici abbiano mostrato risultati migliori rispetto a quelle più complesse, rimane aperta la possibilità che strutture più sofisticate, opportunamente calibrate, possano incrementare ulteriormente le performance predittive.

Alla luce di queste osservazioni, si può affermare che la rete neruale implementata, pur mostrando segnali promettenti, non costituisce una soluzione definitiva per l'ottimizzazione delle campagne PPC. Piuttosto, essa deve essere considerata come un tassello fondamentale di un framework più ampio, in cui tecniche di *deep learning* e algoritmi di decisione sequenziale, come i *contextual bandit*, possono essere combinati per sfruttare i rispettivi punti di forza. In questo scenario, la MLP può svolgere un ruolo centrale nella fase di *warm start*, contribuendo a ridurre il regret iniziale e ad accelerare la convergenza dell'algoritmo verso strategie di bidding più efficienti.

Infine, questa analisi mette in evidenza la necessità di ulteriori sviluppi futuri, sia in termini di arricchimento e rappresentatività dei dati, sia in termini di sperimentazione di architetture neurali avanzate e metodologie di regolarizzazione più efficaci. Tali direzioni di ricerca potranno consolidare il contributo di questa tesi alla letteratura scientifica e, al contempo, fornire strumenti pratici alle aziende che intendono sfruttare l'intelligenza artificiale per rendere le proprie campagne pubblicitarie più competitive ed efficaci.

### 5.2.2 Valutazione complessiva dei modelli di Contextual Bandit

L'analisi condotta sui due modelli di Contextual Bandit sviluppati ha permesso di evidenziare punti di forza e criticità legate alle diverse implementazioni, nonché alle varianti architetturali introdotte (MLP con e senza pesi - `contextualbandit_MLP` e `ContextualBandit_MLP_no_weights`, modelli privi di MLP di supporto - `ContextualBandit_no_MLP` e `ContextualBandit_nw_no_MLP`-). In questa sezione vengono discusse le implicazioni scientifiche e operative dei risultati ottenuti, con l'obiettivo di valutare in che misura ciascun approccio possa essere considerato affidabile e coerente con il contesto applicativo dell'ottimizzazione delle campagne PPC su Amazon.

La prima implementazione del bandit si è rivelata fortemente instabile: il calcolo del *regret*, basato direttamente sui dati reali e sulla funzione descritta in 3.4.3, ha prodotto valori anomali, spesso negativi e di magnitudine elevata. Tali anomalie hanno reso difficile interpretare i risultati, in quanto il modello mostrava prestazioni apparentemente superiori rispetto allo scenario reale di riferimento. In questa versione, le varianti `contextualbandit_MLP` e `ContextualBandit_MLP_no_weights` hanno mantenuto una coerenza parziale nei valori medi di *regret*, ma hanno comunque presentato distribuzioni molto ampie e poco controllabili. Le versioni `ContextualBandit_no_MLP` e `ContextualBandit_nw_no_MLP` hanno mostrato un livello ancora maggiore di variabilità, con picchi di regret estremamente alti, evidenziando una mancanza di stabilità nella capacità del modello di gestire le scelte sequenziali.

In termini di reward, la prima implementazione ha restituito valori medi simili fra le diverse varianti, suggerendo che la capacità di generare reward non fosse influenzata in maniera significativa dalla presenza o assenza di MLP. Tuttavia, la variabilità osservata nel regret ha reso questa versione poco utilizzabile dal punto di vista operativo: il rischio di adottare configurazioni che producono risultati distorti o poco realistici è risultato troppo elevato. In altre parole, la prima implementazione si caratterizza per un trade-off sbilanciato verso il rischio: l'algoritmo è in grado di esplorare scenari estremi, ma lo fa al prezzo di una bassa affidabilità e di una difficile interpretabilità.

La seconda implementazione, basata sulla ridefinizione della funzione di ricompensa come indicato in 3.4.3, ha consentito di ottenere un notevole miglioramento in termini di stabilità e coerenza scientifica. Il calcolo del *regret*, effettuato sulle predizioni stimate dal bandit stesso, ha riportato i risultati entro intervalli plausibili, con valori medi compresi generalmente tra 0% e 40%. Questo ha reso le analisi statistiche più interpretabili e vicine alla realtà del problema.

Le varianti architetturali hanno mostrato dinamiche differenti ma complessivamente più controllate rispetto alla prima implementazione. Le versioni con MLP, sia con che senza pesi, hanno presentato valori di regret comparabili, pur mantenendo una maggiore sensibilità agli iperparametri come il learning rate. La presenza di pesi iniziali ha garantito maggiore stabilità nella fase di apprendimento, riducendo la variabilità dei risultati e migliorando la capacità del modello di convergere verso configurazioni ottimali. La variante priva di MLP ha mostrato un comportamento più costante ma meno performante, con valori medi inferiori di reward e una capacità limitata di esplorare soluzioni con alto potenziale. Infine, la configurazione senza pesi né MLP ha confermato di essere la meno performante, pur distinguendosi per una stabilità intrinseca dovuta alla minore complessità architetturale.

Dal confronto diretto tra le due versioni emerge dunque una distinzione chiara: la prima implementazione rappresenta un approccio ad alto rischio, capace di generare con-

figurazioni estreme e difficilmente interpretabili, mentre la seconda offre un compromesso più sicuro e stabile, sacrificando parzialmente la capacità esplorativa a favore di risultati scientificamente coerenti e operativamente più affidabili. Questo trade-off tra rischio e sicurezza riflette una scelta metodologica cruciale: nella prima versione il modello rischia di sovra-ottimizzare, restituendo scenari non realistici, mentre nella seconda il processo decisionale rimane ancorato a predizioni consistenti, garantendo un maggior controllo delle performance.

In sintesi, i risultati indicano che le configurazioni con MLP e pesi iniziali all'interno della seconda versione del bandit rappresentano l'approccio più equilibrato, poiché offrono un reward medio competitivo, una deviazione standard contenuta e una buona coerenza con le metriche di business. Le altre varianti, pur mostrando caratteristiche interessanti (maggiore stabilità senza MLP, minore varianza senza pesi), non raggiungono lo stesso livello di affidabilità e performance complessiva. La seconda implementazione, quindi, si pone come base solida per futuri sviluppi, consentendo di costruire un framework di ottimizzazione robusto e scientificamente valido per il dominio delle campagne PPC su Amazon.

I problemi riscontrati con entrambe le varianti fanno emergere la difficoltà in tutti i modelli di generalizzare i dati che sono stati utilizzati per l'addestramento. Come già visto nella sezione 5.2.1, possiamo distinguere in due macro aree i problemi rilegati al modello:

- **Problemi legati alla rete**: Come discusso nel capitolo 5.2.1, anche il contextual bandit presenta le stesse criticità derivanti dalla rete neurale che costituisce il *core* del modello (implementata nel costruttore `ContextualBandit.__init__`, vedi eq. (3.4.3)) e dalla MLP di supporto. Tali problematiche coincidono con quelle già evidenziate nella sezione 5.2.1. Nel modello `ContextualBandit_nw_no_MLP` la rete neurale interna corrisponde alla stessa rete neurale utilizzata nella MLP di supporto. Questo *ablation study* è stato condotto per completare lo studio in maniera sistematica; tuttavia, non si esclude che una progettazione più specifica della rete neurale possa condurre a risultati significativamente migliori.

- **Problemi legati al dataset**: Poiché il dataset impiegato per l'addestramento della MLP e del contextual bandit è lo stesso, le criticità già discusse nella sezione 5.2.1 risultano pienamente trasferibili anche a questo modello.

## 5.3 Sviluppi Futuri

In questa sezione si discutono i possibili sviluppi futuri di questo lavoro di ricerca, con l'obiettivo di evidenziare i margini di miglioramento e le potenzialità ancora inesplorate. L'analisi si articola in due direttrici principali: da un lato le migliorie legate alla qualità e alla quantità del dataset disponibile, dall'altro le ottimizzazioni dei modelli di machine learning utilizzati, sia in termini architetturali che metodologici. Tale prospettiva si inserisce all'interno della letteratura scientifica più recente che sottolinea come la qualità del dato e l'adeguata scelta dell'algoritmo rappresentino due pilastri fondamentali per l'ottenimento di risultati solidi e generalizzabili [21, 6, 3].

### 5.3.1 Dataset: prospettive di arricchimento e ottimizzazione

Il dataset utilizzato in questo elaborato ha permesso di ottenere risultati significativi; tuttavia, la sua struttura presenta margini di miglioramento che potrebbero ampliare notevolmente la capacità predittiva dei modelli. Gli sviluppi futuri legati al dataset possono essere suddivisi in più sottocategorie, come di seguito discusso.

#### Codificare le strategie delle campagne

Le campagne pubblicitarie su Amazon non sono tutte progettate con lo stesso obiettivo. Alcune mirano alla massimizzazione del profitto diretto, altre alla pura visibilità del brand, anche a costo di sostenere perdite economiche a breve termine. In letteratura, questa dicotomia è stata spesso descritta come *performance-oriented* versus *awareness-oriented strategies* [31].

Un miglioramento fondamentale del dataset potrebbe consistere nella codifica esplicita delle strategie adottate. Ciò significa aggiungere metadati che specifichino se una determinata campagna è stata concepita per:

- incrementare la quota di mercato tramite bid elevati,
- aumentare la visibilità del brand in contesti specifici,
- promuovere prodotti complementari,
- o ancora testare nuove keyword con budget limitati.

Incorporare queste informazioni nel dataset permetterebbe ai modelli di machine learning di apprendere schemi predittivi più robusti, adattando la funzione obiettivo in base al contesto strategico.

#### Ottenere la posizione del prodotto in ricerca organica

Un secondo asse di miglioramento riguarda l'integrazione delle informazioni sul posizionamento organico del prodotto per una determinata keyword. Amazon, infatti, ordina i risultati organici sulla base di parametri interni, tra cui il volume delle vendite, il tasso di conversione e la soddisfazione del cliente.

La posizione organica ha un impatto diretto sul numero di impression ricevute dall'annuncio, rappresentando un indicatore cruciale della competitività del prodotto. A differenza della pubblicità a pagamento, che comporta un costo per click, il posizionamento organico rappresenta un vantaggio competitivo gratuito e sostenibile nel tempo. Un posizionamento organico è tanto più elevato quanto lo sono gli indicatori (recensioni, opinioni del cliente, tipo di prodotto, competitività del prodotto) ma tra tutti, l'indicatore di maggior importanza sono le unità di prodotto vendute, tanto più alte migliore sarà il posizionamento.

Integrare questa metrica nel dataset consentirebbe di stimare più accuratamente la redditività marginale delle campagne pubblicitarie e di distinguere tra la forza intrinseca del prodotto e l'efficacia delle strategie pubblicitarie.

#### Ottenere indicatori di forza degli altri annunci per una data keyword

Un ulteriore sviluppo riguarda la possibilità di modellare la forza competitiva degli annunci concorrenti. La forza di un brand concorrente potrebbe essere stimata attraverso vari indicatori, quali:

- volume delle vendite,
- numero e qualità delle recensioni,
- posizionamento medio nelle ricerche organiche,
- livello di investimento pubblicitario stimato.

In letteratura, sono stati proposti diversi modelli di valutazione della competitività dei brand basati sull'analisi congiunta di vendite, reputazione online e sentiment dei consumatori [13, 20]. L'integrazione di tali metriche consentirebbe al modello di contestualizzare meglio i dati osservati, introducendo una dimensione relativa rispetto alla concorrenza diretta.

#### Codificare dove l'annuncio deve apparire

La localizzazione dell'annuncio all'interno dell'interfaccia di Amazon è un ulteriore aspetto spesso trascurato, ma di grande rilevanza. Gli annunci possono infatti apparire:

- nella pagina dei risultati di ricerca,
- all'interno della pagina prodotto come suggerimenti complementari,
- o come prodotti simili a quelli visualizzati.

Codificare questa informazione nel dataset permetterebbe di comprendere quale collocazione genera il miglior ritorno sull'investimento ($ROI$) per una determinata categoria di prodotti o keyword. Studi precedenti hanno dimostrato che la posizione dell'annuncio influisce significativamente sul tasso di click e di conversione [19].

#### Ottenere più dati

Infine, un miglioramento trasversale riguarda la quantità di dati a disposizione. L'acquisizione di dataset più ampi e aggiornati garantirebbe una maggiore robustezza statistica, riducendo il rischio di overfitting e consentendo l'addestramento di modelli più complessi. Inoltre, un dataset esteso nel tempo permetterebbe di catturare dinamiche stagionali, variazioni dovute a eventi esterni e tendenze di lungo periodo.

### 5.3.2 Modelli: prospettive di miglioramento

Parallelamente alle migliorie del dataset, anche i modelli di machine learning adottati possono essere oggetto di ulteriori sviluppi. In questa sezione si discutono le prospettive relative alle MLP, ai modelli di contextual bandit e ad altre architetture emergenti.

#### Miglioramenti alle MLP

**Creare reti MLP più complesse** Le MLP utilizzate in questo elaborato si basano su un numero limitato di strati nascosti. Tuttavia, la letteratura suggerisce che architetture più profonde, opportunamente regolarizzate, possano catturare meglio la complessità dei dati [21].

Tra i possibili sviluppi si annoverano:

- l'introduzione di strati addizionali con un numero crescente di neuroni,
- l'uso di funzioni di attivazione avanzate come le *Swish* o le *Mish*,
- l'applicazione di tecniche di normalizzazione come *batch normalization* o *layer normalization*,
- e l'adozione di strategie di regolarizzazione più sofisticate, come il *dropconnect*.

**Creare modelli diversi per derivare le feature** Un'ulteriore prospettiva riguarda la possibilità di utilizzare reti differenti per derivare rappresentazioni alternative delle stesse feature. In letteratura, tale approccio è spesso assimilabile alle tecniche di *ensemble feature learning*, che combinano i punti di forza di architetture eterogenee [16].

Un possibile schema consiste nel progettare più MLP parallele, ciascuna specializzata nell'estrazione di feature da un sottoinsieme del dataset, per poi fonderne le rappresentazioni in un livello superiore. Tale metodologia consente di ridurre il rischio che un'unica architettura sovrastimi o sottostimi la rilevanza di determinate variabili.

#### Miglioramenti ai contextual bandit

Il modello di contextual bandit adottato in questo lavoro rappresenta un primo passo verso un'ottimizzazione dinamica delle campagne pubblicitarie. Tuttavia, diversi miglioramenti sono possibili:

- implementare algoritmi avanzati come *Thompson Sampling with Neural Networks* [43],
- introdurre meccanismi di *uncertainty estimation* tramite reti bayesiane,
- esplorare versioni *contextual combinatorial bandit*, adatte a scenari in cui più annunci devono essere selezionati contemporaneamente.

Questi sviluppi potrebbero rendere il modello più flessibile nell'adattarsi a contesti dinamici e multi-variabili, come le piattaforme pubblicitarie digitali.

### 5.3.3 Altri sviluppi modellistici

#### Testare una implementazione con i Transformers

Una linea di ricerca particolarmente promettente riguarda l'adozione dei modelli basati su Transformers, già affermatisi in domini come l'elaborazione del linguaggio naturale e la visione artificiale [51, 15]. La loro capacità di catturare relazioni a lungo raggio e di modellare interazioni complesse tra feature potrebbe rivelarsi estremamente utile anche nel dominio delle campagne PPC.

Un possibile sviluppo consisterebbe nel pre-addestrare un modello Transformer sulle keyword e sul loro embedding semantico (ad esempio tramite SBERT) e successivamente adattarlo alle specifiche esigenze predittive del contesto pubblicitario. In letteratura, alcuni lavori recenti hanno mostrato che i Transformers possono superare le MLP tradizionali in compiti di regressione complessi [54].

## 5.4 Conclusioni sugli sviluppi futuri

In sintesi, gli sviluppi futuri di questo progetto si articolano su due piani principali: la raccolta di dati più ricchi e strutturati, capaci di rappresentare meglio la complessità del dominio pubblicitario su Amazon, e l'adozione di modelli di machine learning più potenti e sofisticati, in grado di sfruttare appieno tali dati. La sinergia tra questi due aspetti potrà condurre a una significativa evoluzione delle capacità predittive e decisionali degli algoritmi, avvicinando l'obiettivo di campagne pubblicitarie sempre più ottimizzate e data-driven.

# Bibliografia


[1] URL: https://pandas.pydata.org/docs/reference.

[2] Amazon. *CONTRATTO DI AMAZON ADVERTISING*. 2025. URL: https://advertising.amazon.it/terms.

[3] V. Arnold e Steve Sutton. "The theory of technology dominance: Understanding the impact of intelligent decision aids on decision makers' judgments". In: *Advances in Accounting Behavioral Research* 1 (gen. 1998), pp. 175–194.

[4] Peter Auer, Nicolò Cesa-Bianchi e Paul Fischer. "Finite-time Analysis of the Multiarmed Bandit Problem." In: *Mach. Learn.* 47.2-3 (2002), pp. 235–256. URL: http://dblp.uni-trier.de/db/journals/ml/ml47.html#AuerCF02.

[5] Jimmy Ba, Jamie Kiros e Geoffrey Hinton. "Layer Normalization". In: (lug. 2016). DOI: 10.48550/arXiv.1607.06450.

[6] Christopher Bishop. "Pattern Recognition and Machine Learning". In: vol. 16. Gen. 2006, pp. 140–155. DOI: 10.1117/1.2819119.

[7] Piotr Bojanowski et al. "Enriching word vectors with subword information". In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.

[8] L Breiman. "Random Forests". In: *Machine Learning* 45 (ott. 2001), pp. 5–32. DOI: 10.1023/A:1010950718922.

[9] Jason Brownlee. *What is Data Leakage in Machine Learning?* https://machinelearningmastery.com/data-leakage-machine-learning/. Accessed: 2025-06-30. 2020.

[10] Ricardo Campello, Davoud Moulavi e Joerg Sander. "Density-Based Clustering Based on Hierarchical Density Estimates". In: vol. 7819. Apr. 2013, pp. 160–172. ISBN: 978-3-642-37455-5. DOI: 10.1007/978-3-642-37456-2_14.

[11] Olivier Chapelle e Lihong Li. "An Empirical Evaluation of Thompson Sampling". In: *Advances in Neural Information Processing Systems*. A cura di J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper_files/paper/2011/file/e53a0a2978c28872a4505bdb51db06dc-Paper.pdf.

[12] Tianqi Chen e Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, ago. 2016, pp. 785–794. DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

[13] Judith A. Chevalier e Dina Mayzlin. "The Effect of Word of Mouth on Sales: Online Book Reviews". In: *Journal of Marketing Research* 43.3 (2006), pp. 345–354. DOI: 10.1509/jmkr.43.3.345.

[14] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv preprint arXiv:1810.04805* (2019).

[15] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[16] TG Dietterich. “Ensemble methods in machine learning”. In: gen. 2000, pp. 1–15. ISBN: 3-540-67704-6.

[17] Stitch Fix. *Multi-Armed Bandits and the Stitch Fix Experimentation Platform*. https://multithreaded.stitchfix.com/blog/2020/08/05/bandits/. 2020.

[18] GeeksforGeeks. *Multi-armed bandit problem in reinforcement learning*. Lug. 2025. URL: https://www.geeksforgeeks.org/machine-learning/multi-armed-bandit-problem-in-reinforcement-learning/.

[19] Anindya Ghose, Avi Goldfarb e Sang Pil Han. “Estimating Mobile Search and App Discovery Functions: The Role of Relevance, Location, and Time”. In: *Marketing Science* 32.5 (2014), pp. 679–701. DOI: 10.1287/mksc.2013.0798.

[20] Anindya Ghose e Sha Yang. “Estimating the Welfare Effects of Search Engine Optimization: Evidence from Online Retailing”. In: *Management Science* 58.12 (2012), pp. 2055–2071. DOI: 10.1287/mnsc.1120.1542.

[21] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. Book in preparation for MIT Press. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[22] Dalin Guo et al. *Deep Bayesian Bandits: Exploring in Online Personalized Recommendations*. 2020. arXiv: 2008.00727 [cs.LG]. URL: https://arxiv.org/abs/2008.00727.

[23] Shubhankar Gupta. *What is multi-armed Bandit(MAB) testing?: VWO*. Mag. 2025. URL: https://vwo.com/blog/multi-armed-bandit-algorithm/.

[24] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (set. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[25] Peter J. Huber. “Robust Estimation of a Location Parameter”. In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. DOI: 10.1214/aoms/1177703732. URL: https://doi.org/10.1214/aoms/1177703732.

[26] huggingface. *Sentence-transformers/all-minilm-L6-V2 · hugging face*. Ago. 2021. URL: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

[27] Sergey Ioffe e Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

[28] Hong Kao e Ian H. Witten. “Data Leakage in Data Mining: A Review”. In: *ACM SIGKDD Explorations Newsletter* 20.2 (2018), pp. 30–35. DOI: 10.1145/3291285.3291290.

[29] KDnuggets. *Introduction to Multi-Armed Bandit Problems*. https://www.kdnuggets.com/2023/01/introduction-multiarmed-bandit-problems.html. 2023.

[30] Guolin Ke et al. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree”. In: *Advances in Neural Information Processing Systems*. A cura di I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[31] P. Kotler, K.L. Keller e A. Chernev. *Marketing Management*. Pearson Education, 2021. ISBN: 9781292404813. URL: https://books.google.it/books?id=iTPTzgEACAAJ.

[32] Tor Lattimore e Csaba Szepesvari. “Bandit Algorithms”. In: (2017). URL: https://tor-lattimore.com/downloads/book/book.pdf.

[33] Lihong Li et al. “A contextual-bandit approach to personalized news article recommendation.” In: *WWW*. A cura di Michael Rappa et al. ACM, 2010, pp. 661–670. ISBN: 978-1-60558-799-8. URL: http://dblp.uni-trier.de/db/conf/www/www2010.html#LiCLS10.

[34] Leland McInnes, John Healy e James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2020. arXiv: 1802.03426 [stat.ML]. URL: https://arxiv.org/abs/1802.03426.

[35] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[36] Vinod Nair e Geoffrey Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair”. In: vol. 27. Giu. 2010, pp. 807–814.

[37] OfferFit. *Love multi-armed bandits? Meet their smarter cousins*. https://www.offerfit.ai/content/blog-post/love-multi-armed-bandits-meet-their-smarter-cousins. 2024.

[38] Optimizely. *What is a multi-armed bandit?* ://www.optimizely.com/optimization-glossary/multi-armed-bandit/. 2022.

[39] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[40] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[41] Jeffrey Pennington, Richard Socher e Christopher D Manning. “GloVe: Global Vectors for Word Representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[42] Nils Reimers e Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 2019, pp. 3982–3992.

[43] Carlos Riquelme, George Tucker e Jasper Snoek. “Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling”. In: *International Conference on Learning Representations (ICLR)*. 2018. URL: https://openreview.net/forum?id=SyYe6k-CW.

[44] Herbert Robbins. “Some aspects of the sequential design of experiments”. In: *Bulletin of the American Mathematical Society* 58.5 (1952), pp. 527–535.

[45] Towards Data Science. *An Overview of Contextual Bandits*. https://towardsdatascience.com/an-overview-of-contextual-bandits-53ac3aa45034/. 2024.

[46] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (giu. 2014), pp. 1929–1958.

[47] Counting Stuff. *Whatever happened to the multi-armed bandit?* https://www.counting-stuff.com/whatever-happened-to-the-multi-armed-bandit/. 2024.

[48] Richard S. Sutton e Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[49] WILLIAM R THOMPSON. “ON THE LIKELIHOOD THAT ONE UNKNOWN PROBABILITY EXCEEDS ANOTHER IN VIEW OF THE EVIDENCE OF TWO SAMPLES”. In: *Biometrika* 25.3-4 (dic. 1933), pp. 285–294. ISSN: 0006-3444. DOI: 10.1093/biomet/25.3-4.285. eprint: https://academic.oup.com/biomet/article-pdf/25/3-4/285/513725/25-3-4-285.pdf. URL: https://doi.org/10.1093/biomet/25.3-4.285.

[50] Udemy. *Building a Multi-Armed Bandit System from the Ground Up*. https://medium.com/udemy-engineering/building-a-multi-armed-bandit-system-from-the-ground-up-a-recommendations-and-ranking-case-study-b598f1f880e1. 2022.

[51] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[52] Wenhui Wang et al. “MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers”. In: *arXiv preprint arXiv:2002.10957* (2020). URL: https://arxiv.org/abs/2002.10957.

[53] Wikipedia contributors. *Multi-armed bandit — Wikipedia, The Free Encyclopedia*. [Online; accessed 3-September-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=Multi-armed_bandit&oldid=1306531721.

[54] Min Zeng et al. “Zeng et al.2023”. In: (lug. 2023).