Università degli Studi di Torino

Scuola di Scienze della Natura

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

Tesi triennale

TIME FILTERING

Relatore
Prof. Marino SEGNAN

Candidato
Fabrizio Sanino
925515

Anno Accademico 2021–2022

Un uomo che osa sprecare anche solo un'ora del suo tempo non ha scoperto il valore della vita

---

Charles Darwin

Dichiarazione di originalità

Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata.

Indice

Capitolo 1

Introduzione

1.1 Problemi di ricerca sulle piattaforme di E-commerce

Al giorno d'oggi l'acquisto e la vendita di prodotti sulla rete è una pratica ampiamente utilizzata da parte delle persone. Esistono molte piattaforme che permettono di comprare un qualsiasi tipo di oggetto on-line. Siccome queste aziende presentano sul mercato un'offerta molto ampia, l'utente deve cercare il prodotto che vuole acquistare tra tantissimi altri, deve quindi compiere un'operazione di filtraggio. Questa operazione, teoricamente, dovrebbe essere fatta dai motori di ricerca all'interno degli **E-commerce**. Spesso non è così... all'utente vengono mostrati i prodotti che vengono acquistati più frequentemente, quelli sponsorizzati, quelli che l'azienda vuole vendergli. Infatti, i motori di ricerca, quando l'utente inserisce un determinato prodotto con delle caratteristiche precise, non risponde con solo prodotti aventi quelle caratteristiche ma risponde in base ai gusti generali degli utenti. Questo porta quindi l'utente a dover filtrare i risultati di un filtraggio, consumando spesso moltissimo tempo perchè l'uomo è più lento di un calcolatore.

Facendo una ricerca su un qualsiasi sito di E-commerce possiamo osservare questo fenomeno. (Il prodotto che cerchiamo è arbitrario)

1.1.1 Esempio ricerca

Proviamo a cercare, a titolo di esempio, su Amazon un Notebook ASUS con 16GB di RAM. Questo è il risultato

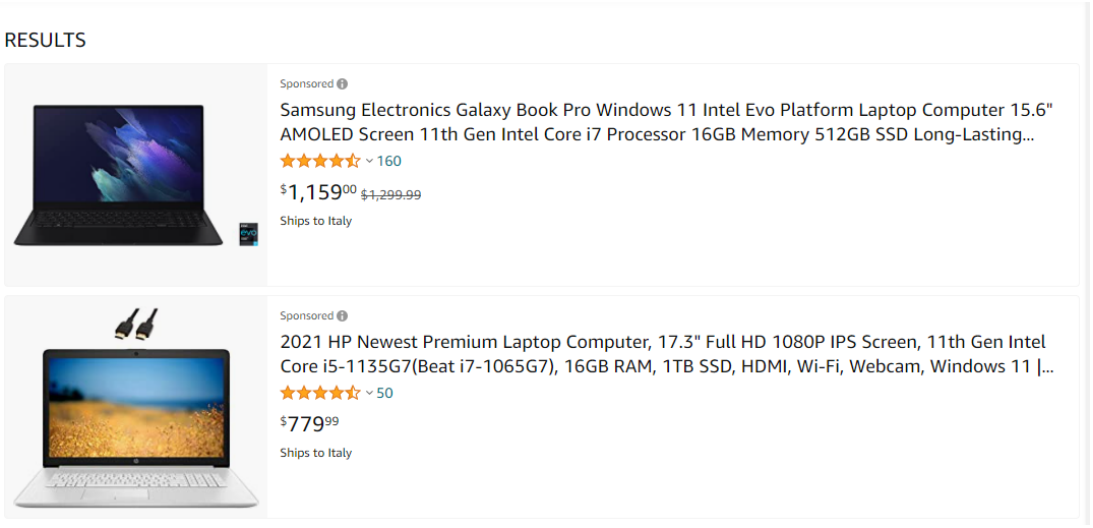

Figura 1.1: Esempio di ricerca

Ci vengono mostrati, come primi due risultati, prodotti che non sono nemmeno della marca corretta, infatti, il primo è un Samsung mentre il secondo un HP. Dopo una serie di ricerche, si è stimato quanto segue:

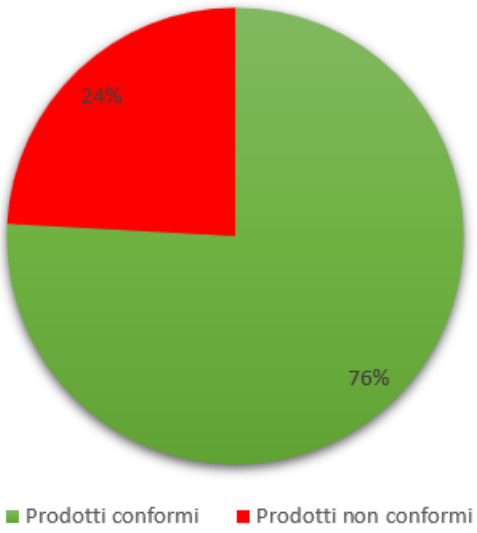

Figura 1.2: Confronto prodotti

cioè in ogni ricerca che l'utente effettua, ogni circa 4 prodotti, è presente un prodotto che non rispecchia le caratteristiche. Più la ricerca diventa precisa, più questa probabilità aumenta.

In questa tesi si è discussa una soluzione di come può essere risolto il problema sopra descritto.

1.2 Soluzione

Per risolvere il problema ho sviluppato un'applicazione Android che permette di filtrare ciò che la ricerca su un qualsiasi sito di E-commerce fornirebbe agli utenti, ovvero non mostrando tutti quei prodotti che non sono conformi rispetto alle specifiche dettate dall'utente. Inoltre l'applicazione permette, tramite una Supervised Machine Learning di classificare il prodotto in base alle recensioni di questo, fornendo all'utente la percentuale di recensioni positive e negative.

Nell'applicazione da me creata è stato fornito un esempio di filtraggio sulla piattaforma di E-commerce Amazon.

1.2.1 Diagramma di contesto

In questo diagramma è rappresentato, in modo sintetico, le relazioni che sono presenti tra i vari sistemi utilizzati e l'utente.

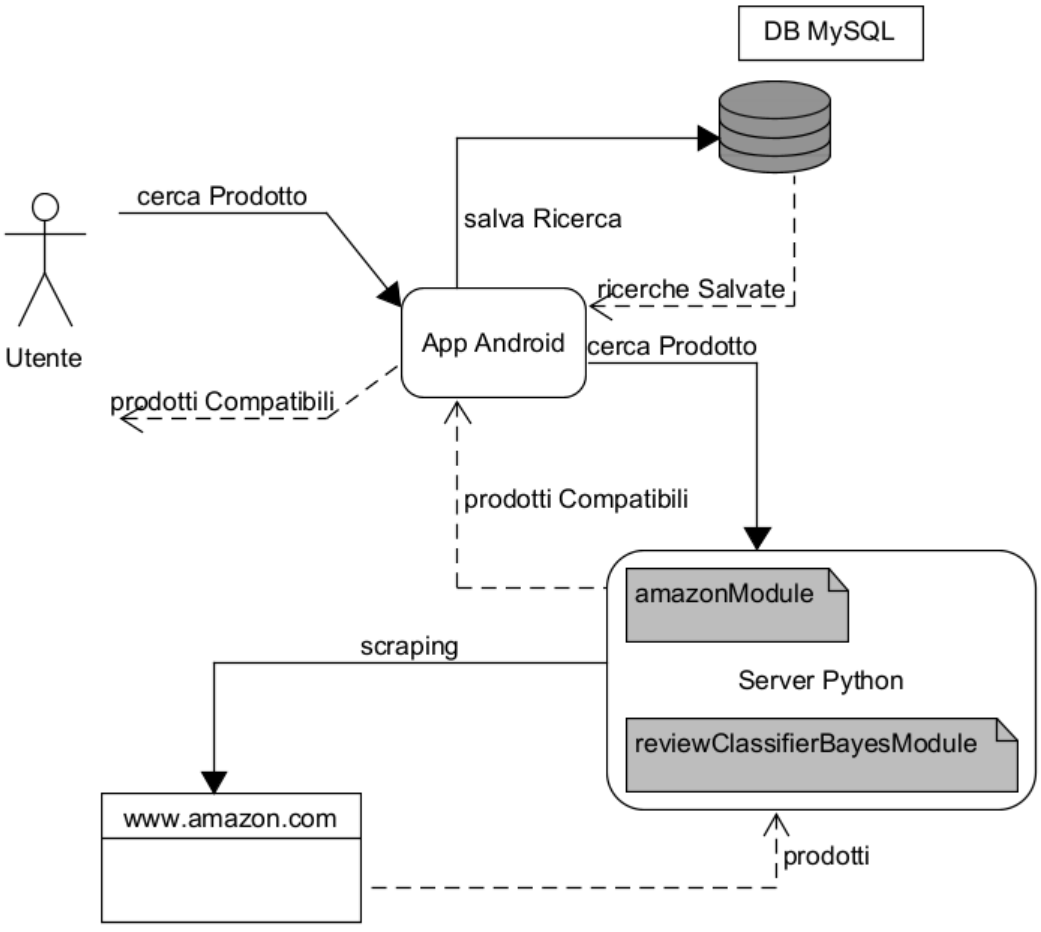

Figura 1.3: Diagramma di contesto

L'utente, utilizzando l'applicazione, inserisce il prodotto che vuole cercare (invece che cercarlo direttamente sul sito di E-commerce). Essa si collega ad un server **Python** che è funziona grazie a due moduli:

- **amazonModule**: si collega al sito di E-commerce permettendo di effettuare lo scraping dei prodotti risultanti da una ricerca.
- **reviewClassifierBayesModule**: permette di analizzare le recensioni di un prodotto e di restituire l'analisi effettuata.

Capitolo 2

Raccolta dei requisiti

2.1 Requisiti utente

In questa sezione sono descritti, tralasciando i dettagli implementativi, i servizi che il sistema offre. Questo passo è stato fatto prima di passare alla fase di sviluppo vero e proprio in modo tale da aver ben chiaro quali fossero gli obbietti che l'utente voleva raggiungere prima di passare ad implementarli.

Dopo un attento e dettagliato studio, i requisiti individuati sono:

- Consentire all'utente di effettuare una ricerca su Amazon tramite un'apposita barra di ricerca.
- Filtraggio dei risultati considerando in AND tutte le parole chiave inserite dall'utente nella ricerca. Un prodotto sarà mostrato all'utente SE E SOLO SE contiene tutte le parole chiavi.
- Salvataggio ed eliminazione di una ricerca effettuata in modo da averla sempre a disposizione.
- Consentire all'utente di ordinare i prodotti in base a vari tipi di ordinamento.
- Consentire all'utente di ottenere una valutazione del prodotto in base alle recensioni di esso.

2.2 Casi d'uso

In questa sezione sono descritti gli scenari interessanti che il sistema software dovrà implementare. Per la stesura è stato deciso di utilizzare il formato breve.

Effettua ricerca

L'utente inserisce il prodotto e le caratteristiche che il prodotto deve avere all'interno di una barra di ricerca. Il sistema filtra i risultati prodotti dal sito di E-commerce e presenta la lista all'utente. L'utente consulta i risultati e naviga sulla pagina di un prodotto.

Salva ricerca

L'utente preme sul pulsante per salvare la ricerca. Il sistema carica su un sistema di memorizzazione la ricerca e conferma al cliente che l'operazione è avvenuta.

Ordina prodotti

L'utente preme sul pulsante per ordinare i prodotti decidendo l'ordine che essi devono avere. Il sistema ordina i prodotti secondo il criterio deciso dall'utente e mostra il risultato.

Valutazione prodotto

L'utente preme sul pulsante per valutare il prodotto. Il sistema consulta la recensione e decide la valutazione da assegnare al prodotto mostrandola all'utente.

Dopo aver analizzato nel dettaglio ciascuno caso d'uso, sono passato alla rappresentazione, tramite i *Diagrammi di sequenza (**SSD**)*, di ciascun caso d'uso in modo da evidenziare tutte le azioni che l'utente può fare e le eventuali eccezioni scatenate dal sistema.

Per brevità verrà riportato un solo diagramma di sequenza riferito al caso d'uso "Effettua ricerca" che rappresenta lo scenario principale di successo.

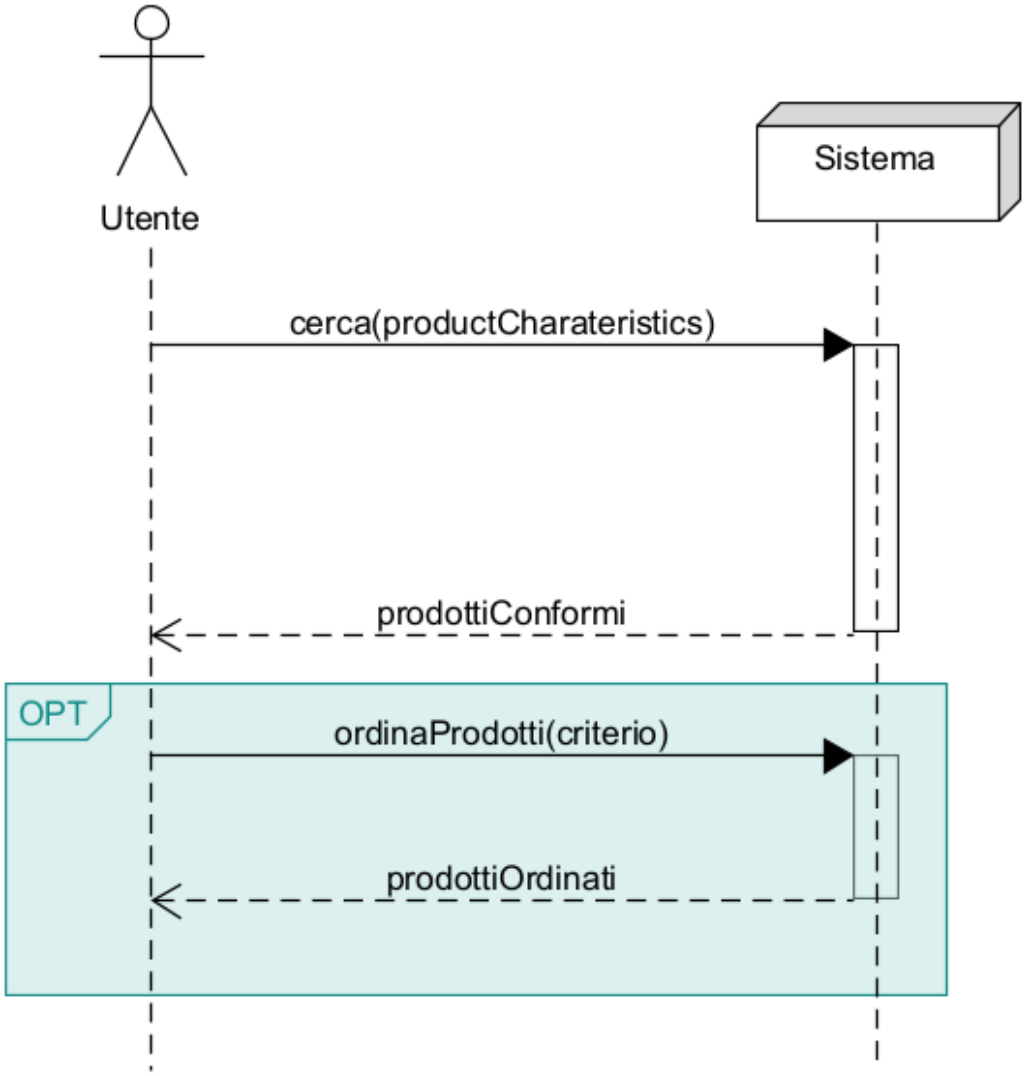

Figura 2.1: Diagramma di sequenza "Effettua ricerca"

Capitolo 3

Architettura

3.1 Front-end

Per sviluppare i casi d'uso sopra citati, è stata realizzata un'applicazione per il sistema operativo **Android** in cui l'utente può facilmente effettuare la proprie ricerche senza dover utilizzare direttamente il sito di E-commerce.

L'applicazione è organizzata mediante l'utilizzo del pattern **MVVM**.

3.1.1 MVVM

E' un **Pattern Architetturale**. Il pattern è una *Soluzione progettuale generale ad un problema ricorrente*. Nello specifico, MVVM, è uno schema che permette di organizzare la struttura di un sistema software suddividendo in modo chiaro le parti che lo compongono. In MVVM sono presenti 3 elementi fondamentali:

- **Model**: in cui sono presenti i dati che utilizza l'applicazione.
- **View**: contiene l'interfaccia utente, che è formata da 2 Fragment (Home e Saved searches).
- **ViewModel**: fa da link tra il Model e le View ed è utilizzato per manipolare i dati che arrivano dal Server.

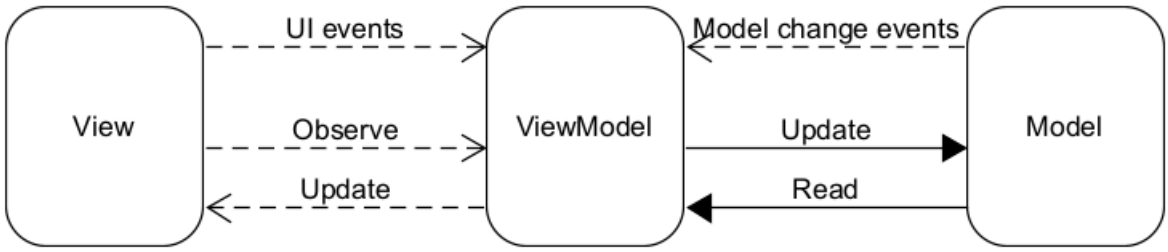

Figura 3.1: Pattern MVVM

3.1.2 Android

E' un sistema operativo **open source** basato su **Linux** che viene eseguito su smartphone, tablet e computer. Offre un approccio unificato nello sviluppo di applicazioni per i dispositivi mobili cioè gli sviluppatori possono sviluppare solo applicazioni per Android e queste verranno eseguite solo su dispositivi Android. Per lo sviluppo di applicazioni si utilizza spesso **Java** ma è anche possibile utilizzare **Kotlin**.

Figura 3.2: Logo dell'applicazione

3.2 Back-end

L'applicazione si collega ad un server **Python** il cui compito principale è quello di navigare sul sito di E-commerce, effettuare una ricerca del prodotto richiesto dall'utente e filtrare i risultati prodotti in modo tale che i prodotti restituiti siano il più conformi possibili a quelle che sono state le richieste dell'utente.

Per poter analizzare i prodotti che un E-commerce restituisce, viene utilizzato il **Web Scraping** (Capitolo 5.2). Ogni sito di E-commerce ha una sua struttura, quindi lo Scraping deve essere effettuato "ad hoc" per ciascuno. A titolo di esempio è stato effettuato per Amazon ma può essere fatto per qualsiasi. Per modificare il sito di E-commerce su cui l'applicazione lavora è possibile definire un apposita classe (con il nome *sitoDiE-commerce*Module). La classe creata deve necessariamente implementare tutti i metodi che sono predenti nella classe astratta *scarpingModuleAbstract* in modo tale che il server, e quindi il filtraggio, possa continuare a funzionare.

La classificazione dei prodotti, ovvero la raccolta delle recensioni e l'analisi di esse, è fatta in un modulo (*reviewClassifierBayes*) utilizzando il **Classificatore Bayesiano**. Siccome ci sono più tecniche utilizzabili, il classificatore è definito su un modulo a parte in modo tale che possa essere cambiato con un altro tipo di classificatore solo modificando l'*import* sul server. Siccome, per funzionare, il classificatore ha bisogno delle recensioni, esse devono essere ottenute tramite lo Scraping che deve essere fatto dal modulo di Scraping costruito appositamente per il sito di E-commerce.

Grazie a questa architettura, modificando un numero limitato di linee di codice, è possibile cambiare il sito di E-commerce (Amazom, AliExpress, Ebay, ...) su cui fare il filtraggio e il tipo di classificatore usato stimare la positività o la negatività di una recensione.

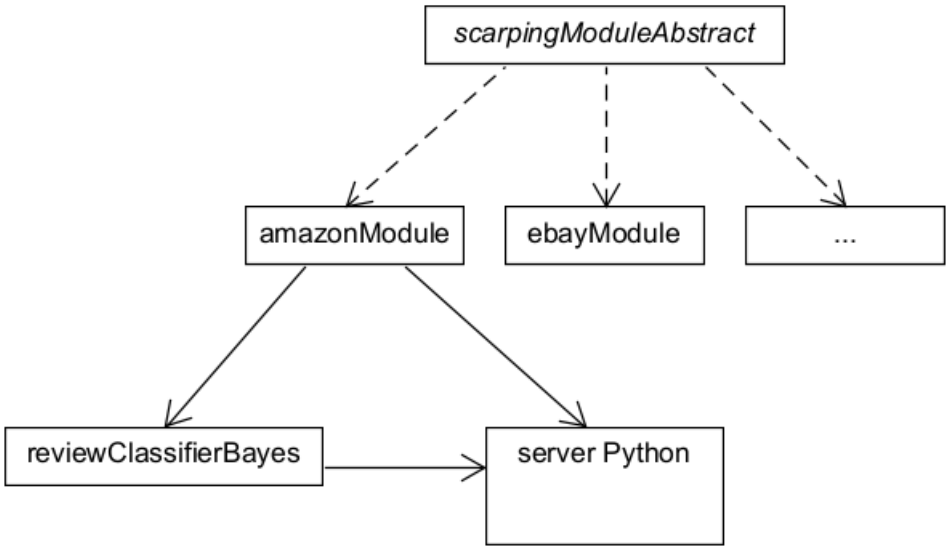

Figura 3.3: Architettura Server

Capitolo 4

Front-end

In questo capitolo verrà descritta l'interfaccia grafica dell'applicazione creata, chiamata **TIME FILTERING**.

4.1 Home - Saved searches

La prima schermata che l'utente vede quando avvia l'applicazione è la HOME. Nella HOME (Figura 4.1, Pagina 15) l'utente trova una casella di ricerca in cui può inserire il prodotto, e le caratteristiche di cui è interessato, che vuole cercare. L'applicazione si collegherà ad un Server che restituirà i risultati, filtrati secondo le specifiche (Figura 4.2, Pagina 15).

Accanto al prodotto compare una piccola immagine che indica l'affidabilità del prodotto, ovvero quanto è pertinente rispetto alle specifiche inserite nella ricerca. L'affidabilità e decisa dal server (Sezione 5.2.4).

Assieme al prodotto compare anche un pulsante CLASSIFY che permette di valutarne le recensioni. A valutazione completata comparirà una Dialog con indicate le percentuali di recensioni positive, negative e neutrali (Figura 4.4, Pagina 15). In questo modo l'utente, senza aver nemmeno controllato il prodotto, può aver una valutazione di esso facendogli quindi risparmiare tempo, evitando così quei prodotti giudicati "scadenti" dagli altri utenti.

Utilizzando il menu, in alto a destra, l'utente può ordinare i risultati in base al prezzo, all'affidabilità e al rating basato sulle stelle inserite dall'utente durante la scrittura della recensione.

Aprendo il menu, in alto a sinistra, è possibile navigare sulla schermata SAVED SEARCHES. Nella SAVED SEARCHES (Figura 4.3, Pagina 15) l'utente troverà le ricerche salvate, tramite il tasto SAVE (Figura 4.2, Pagina 15) dopo aver fatto una ricerca nella HOME.

Utilizzando il tasto VIEW può mostrare tutti i prodotti che fanno parte di quella ricerca mentre tramite il tasto DELETE può eliminare la ricerca. In questo modo l'utente può facilmente ed in poco tempo, accedere alle sue ricerche preferite.

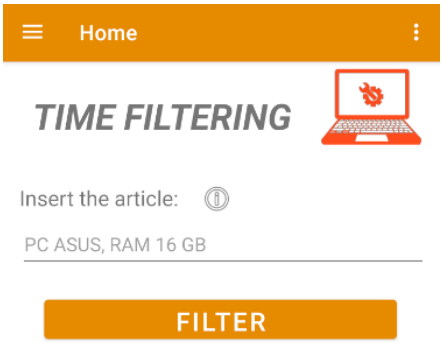

Figura 4.1: Home

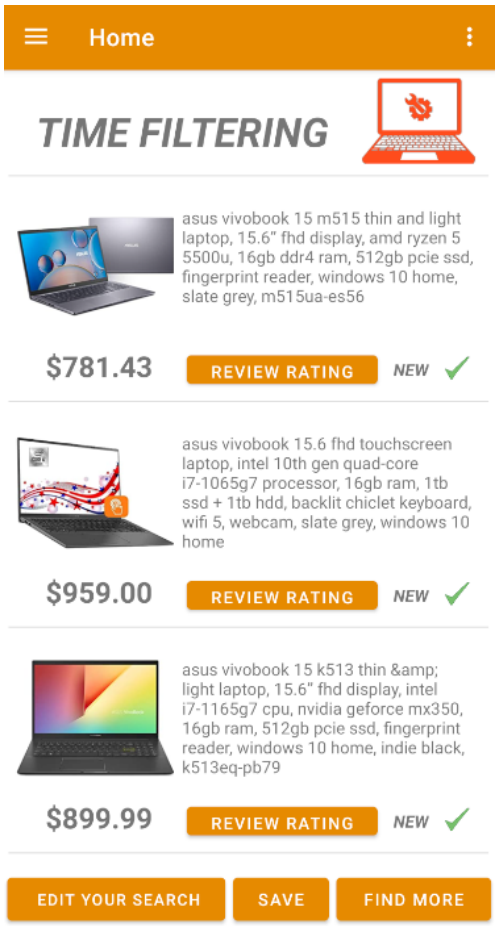

Figura 4.2: Lista dei prodotti

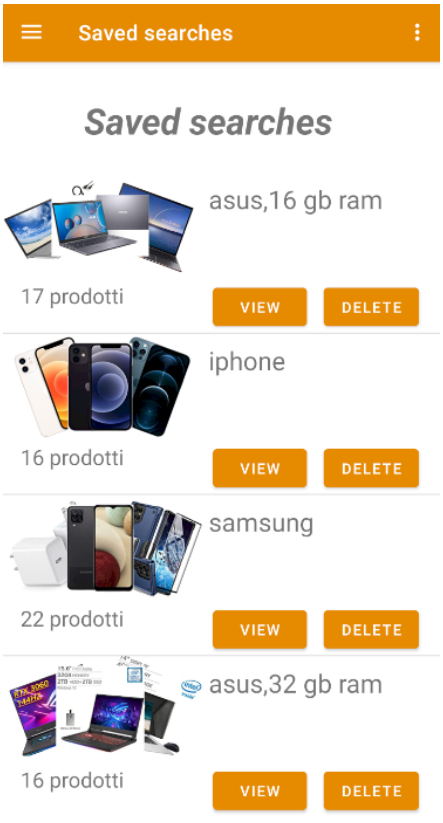

Figura 4.3: Ricerche salvate

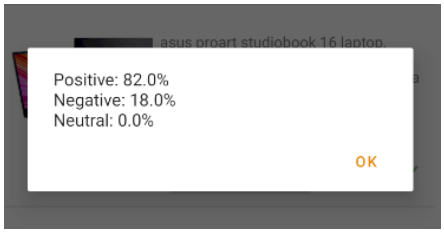

Figura 4.4: Classifica recensioni

4.2 Specifiche tecniche

L'applicazione è stata realizzata usando **JAVA** (Andorid nativo).E' organizzata mediante l'utilizzo del pattern **MVVM** spiegato in precedenza (Capitolo 3.1).

Il ViewModel è la parte fondamentale dell'applicazione. E' rappresentato con la classe **NetworkViewModel**. Ha vari metodi che vengono richiamati dall'interfaccia grafica per andare a reperire dei dati. Una volta che il Server ha risposto, i dati vengono manipolati per poter essere rappresentati nella View e tramite i **LiveData** viene notificata alla View che l'operazione è stata completata.

I LiveData vengono utilizzati per osservare le modifiche di un particolare viewModel e quindi aggiornare l'interfaccia grafica quando avviene la modifica. Essi sono sensibili al ciclo di vita dell'applicazione ovvero ogni volta che un LiveData viene modificato, gli aggiornamenti vengono inviati solo a quelle componenti dell'app che sono nello stato *Attivo*. Tutte quelle componenti che non sono attive verranno notificate non appena lo diventeranno. Qui sotto è riportata una parte di codice in cui vengono utilizzati i LiveData. La funzione, in background, contatta il server e ottiene tutte le ricerche che l'utente ha salvato.

```
public class findResearchDetails extends AsyncTask<JSONObject, Void,
    JSONArray> {
        @Override
        protected JSONArray doInBackground(JSONObject... jsonObjects) {
            return
    sentPOSTRequest(urls.getServerUrlGetResearchDetail(),
    jsonObjects[0]);
        }

        @Override
        protected void onPostExecute(JSONArray result) {
            searchResults.emptyList();

            JSONObject retValue = new JSONObject();

            try {
                if (result.getJSONObject(0).getBoolean("error")) {
                    retValue.put("error", true);
                    retValue.put("errorDescription",
    result.getJSONObject(0).getString("errorDescription"));


    researchDetailsLiveData.updateResearchDetails(retValue);
                } else {
                    for (int i = 1; i < result.length(); i++) {
                        JSONObject item = result.getJSONObject(i);
                        SearchResult s = new
    SearchResult(item.getString("productName"),
    item.getString("productLink"), item.getString("productImg"),
```

```
item.getString("price"), item.getInt("reliability"), false,
Float.parseFloat(item.getString("ratingReview")));
                    searchResults.addElement(s);
                }

                searchResults.getArrayList().sort(Comparator
                .comparingInt(SearchResult::getReliability));
                Collections.reverse(searchResults.getArrayList());

adapterSearch.setData(searchResults.getArrayList());

                searchResults.setSavedSearch(true);

                retValue.put("error", false);


researchDetailsLiveData.updateResearchDetails(retValue);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}
```

Con l'istruzione

```
researchDetailsLiveData.updateResearchDetails(retValue);
```

notifichiamo alla View che le ricerche salvate sono arrivate o se ci sono stati degli errori nel download.

La View tramite

```
networkViewModel.getReadResearchLiveData()
    .observe(getViewLifecycleOwner(), result -> {...});
```

riceve la notifica del cambiamento di stato ed effettua le operazioni necessarie. In questo caso, se non sono avvenuti degli errori, nasconde la dialog del caricamento o altrimenti, in caso di errore, notifica all'utente il tipo di errore.

Capitolo 5

Back-end

5.1 Server

L'applicazione comunica con un Server scritto in **Python**. Si è deciso quel linguaggio in quanto, per andare ad analizzare i risultati di una ricerca sul Web è necessario utilizzare lo **Scraping** ed il modo più veloce per effettuarlo è tramite Python.

5.1.1 Python

E' un linguaggio di programmazione avente le seguenti caratteristiche:

- **interpretato**: ovvero è processato a tempo di esecuzione tramite l'interprete, in questo modo, il programma, non deve essere compilato prima della sua esecuzione.
- **orientato agli oggetti**: cioè supporta lo stile di programmazione in cui il codice è incapsulato in oggetti.
- **interattivo**: è possibile dialogare direttamente con l'interprete.

Python è molto utile quando è necessaria una rapidità nello sviluppo delle applicazioni ma soprattutto quando è necessario connettere delle componenti tra di loro, infatti, proprio per questo motivo è stato utilizzato. Supporta moduli e package incoraggiando così la programmazione modulare ed il riutilizzo di codice.

5.1.2 Flask

Flask è un web framework che permette di sviluppare applicazioni web in modo facile e veloce. Con una numero molto piccolo di istruzioni permette di creare un server. Nello specifico, Flask, è un micro-framework ovvero ha poca dipendenza, talvolta anche senza dipendenze, dalle librerie esterne. In questo modo risulta molto più veloce ed efficiente.

Ecco un esempio di come è stata creata una "route" per rendere disponibile all'esterno un servizio:

```
from flask import Flask
from flask import request
from flask import jsonify

@app.route("/deleteResearch", methods=['POST'])
def deleteResearch():
    content_type = request.headers.get('Content-Type')
    if content_type != 'application/json;charset=UTF-8':
        return {"error": "Content-Type not supported!"}, 422
    else:
        clientParameter = json.loads(request.get_data(), strict=False)
        idResearch = clientParameter["idResearch"]

        try:
            connection = mysql.connector.connect(user='root',
   password='', host='127.0.0.1', database='savedResearch')

            if connection.is_connected():
                cursor = connection.cursor()

                try:
                    #Elimino tutti i prodotti della ricerca
                    cursor.execute("DELETE FROM savedproducts WHERE
   researchNum = " + str(idResearch))

                    connection.commit()

                    #Elimino la ricerca
                    cursor.execute("DELETE FROM search WHERE id = " +
   str(idResearch))

                    connection.commit()

                    connection.close()

                    return jsonify([{
                        "error":False
                    }])
                except:
                    if connection.is_connected():
                        cursor.close()
                        connection.close()

                    return jsonify([{
                        "error": True,
                        "errorDescription": "Error while deleting"
                    }])
```

```
        except:
            return jsonify([{
                "error": True,
                "errorDescription": "Connection error"
            }])

if __name__ == "__main__":
    app.run()
```

In questo modo, collegandoci alla URL `127.0.0.1/deleteResearch` viene richiamato la funzione `deleteResearch` che permette di servire la richiesta. In questo caso, viene eliminata la ricerca salvata sul Database.

5.1.3 Database

Per salvare le ricerche effettuate dall'utente ho utilizzato un Databese SQL-based, nel caso specifico MySQL. La connessione con il Database server avviene utilizzando un connector. Esso serve per permettere di interagire con il Database server definendo un'interfaccia standard a cui tutti i Database per Python devono conformarsi.

Una volta creata la connessione, tramite la creazione di un *cursor* è possibile inviare delle query al Databse. Quando sono finite le modifiche è necessario che venga fatto `connection.commit()` che permette di rendere permanenti le modifiche e terminare la transazione. Nel caso in cui sia accaduto un errore, tramite `connection.rollback()` lo stato del database viene riportato a quello precedente alle modifiche effettuate nella transazione corrente annullando così le modifiche.

5.2 Scraping

Per poter analizzare i prodotti forniti da Amazon durante la ricerca, è stato necessario utilizzare la tecnica dello Web Scraping.

Il Web Scraping è un processo che permette di estrarre contenuti e dati da una pagina web. E' possibile estrarre i contenuti in più formati, il formato che ho utilizzato è l'HTML. Il processo può naturalmente essere fatto manualmente andando ad analizzare l'HTML della pagina ed estraendo i dati "a mano", ma lo scopo del Web Scarping è proprio quello di automatizzare il processo ed aumentare la capacità di dati che possono essere collezionati. Quando ci si approccia allo Scraping bisogna fare attenzione a due aspetti molto importanti:

- **Varietà**: Ogni sito web è costruito in modo diverso, ha una specifica struttura HTML. Quindi, per ogni sito di cui vogliamo collezionare i dati, è necessario effettuare un lavoro "ad-hoc" ripetendo tutti i passaggi dall'inizio. Per ovviare a questo problema ho diviso la parte server dalla parte di Scraping in modo tale che il server sia riutilizzabile (Capitolo 3.2).

- **Durabilità**: I siti web spesso vengono aggiornati, ovvero viene modificata la loro struttura e quindi il codice HTML che li compone. Questo porta quindi a dover aggiornare di frequente il tool di Scapring perchè un giorno può funzionare, ma il giorno dopo no. Quindi saranno necessarie integrazioni durante tutta la vita della nostra applicazione.

Un'alternativa allo scraping sono le API. Esse svolgono la funzione di intermediari cioè permettono a due applicazioni di poter comunicare. La comunicazione avviene tramite lo scambio di JSON o di XML che sono dei formati per lo scambio di dati strutturati. In questo modo non è necessario convertire l'HTML in dati strutturati perchè lo sono già. Questa tecnica però non è sempre utilizzabile, infatti Amazon fornisce solo API a pagamento.

5.2.1 Analisi struttura HTML

Prima di passare allo Scraping è necessario analizzare la pagina per capire come essa è strutturata in modo da creare delle procedure automatizzate che riescano ad ottenere le informazioni necessarie. Per prima cosa ho analizzato la pagina HTML che viene restituita dopo una ricerca su Amazon.
(Di seguito un esempio semplificato di come è formata)

```
<div data-component-type="s-search-result">
    <span class="a-size-base-plus a-color-base a-text-normal">
        <!-- Nome del prodotto -->
    </span>
    <span class="a-offscreen">
        <!-- Prezzo del prodotto -->
    </span>
    <a class="a-link-normal s-underline-text s-underline-link-text
   s-link-style a-text-normal">
        <!-- Link all'immagine del prodotto -->
    </a>
</div>
```

Dopo di che ho analizzato la pagina di un po' di prodotti per capire quali fossero le regolarità dei tag utilizzati.
(Di seguito un esempio semplificato di come sono formati)

```
<div id="featurebullets_feature_div">
    <ul>
        <li>
            <span class="a-list-item">
                <!-- Caratteristica del prodotto -->
            </span>
        </li>
        ...
        ...
    </ul>
```

```
</div>
<div id="productDescription">
    <p>
        <!-- Descrizione del prodotto -->
    </p>
</div>
<div>
    <table id="productDetails_techSpec_section_1">
        <tbody>
            <tr>
                <th>
                    <!-- Titolo dettaglio -->
                </th>
                <td>
                    <!-- Dettaglio -->
                </td>
            </tr>
            ...
            ...
        </tbody>
    </table>
</div>
```

Una volta analizzata la pagina bisogna decidere che tipo di libreria utilizzare per trasformare i dati contenuti nell'HTML in modo che possano essere utilizzati. Per prima cosa è stato deciso di utilizzare **Beautiful Soup**.

5.2.2 Beautiful Soup

E' una libreria Python che permette di convertire i dati non strutturati contenuti nell'HTML in dati strutturati che possano essere utilizzati.

Tramite la seguente istruzione

```
soup = BeautifulSoup(page.content, "html.parser")
```

la libreria scarica il contenuto HTML nella variabile `soup` e rende possibile la navigazione di esso.

Il problema di questa tecnica è stata la lentezza, infatti, dopo una serie di test, è stato deciso di abbandonare questa libreria per adottarne un'altra molto più performante. Si è, quindi, deciso di utilizzare **Selenium**.

5.2.3 Selenium

E' una libreria Python che permette di emulare l'iterazione umana su un browser senza che effettivamente essa venga fatta da un umano ma bensì da uno strumento automatico. E' quindi possibile utilizzare un browser guidandolo con del codice, nel mio caso Python, per reperire delle informazioni sulla rete. Per far funzionare Selenium è necessario scaricare un **Web Driver**. Esso serve per

aprire in modo automatico il browser che decidiamo di utilizzare e di accedere al sito web che vogliamo. Naturalmente è necessario scaricare il Web Driver specifico per il browser che abbiamo intenzione di utilizzare. Nel mio caso ho utilizzato Chrome in quanto, dopo una serie di test, si è rivelato quello più performante con questa libreria.

Dopo aver fatto ciò, per prima cosa bisogna collegarsi alla pagina principale di Amazon. Dopo essersi collegati si deve inserire nella barra di ricerca il nome del prodotto che ha cercato l'utente ed inviare il comando "ENTER". Dopo di che, quando sono arrivati i risultati, si devono andare a leggere in modo da poterli analizzare. Tutto questo viene fatto con dei comandi di libreria.

(Di seguito una versione semplificata del codice utilizzato)

```
linkAmazon = 'https://www.amazon.com'
mainDriver.get(linkAmazon)

mainDriver.implicitly_wait(1)
search = mainDriver.find_element(By.ID, 'twotabsearchtextbox')
search.send_keys(keywordProductName.replace(',', ' '))
search.send_keys(Keys.ENTER)

mainDriver.implicitly_wait(0.5)

items = wait(mainDriver, 10).until(
        EC.presence_of_all_elements_located((By.XPATH,
    '//div[contains(@data-component-type, "s-search-result")]')))
```

Siccome la connessione non è veloce tanto quando lo scraping è necessario inserire delle `sleep(...)` o `driver.implicitly_wait(...)` in modo tale da lasciare il tempo alla connessione di scaricare la pagina. Inoltre non è possibile solo analizzare il titolo del prodotto, perchè potrebbero non essere presenti dei dettagli importanti, quindi bisogna navigare nella pagina del prodotto ed analizzare anche quella. Questo però porta al rallentamento delle prestazioni complessive, ovvero ho stimato che per analizzare 25 prodotti (quelli contenuti in una pagina Amazon) ci vogliono circa 3 minuti (in base a quanto è specifico il prodotto da cercare).

Una volta ottenuti i prodotti risultanti dalla ricerca su Amazon è necessario filtrarli in base alle caratteristiche che ha richiesto l'utente.

5.2.4 Filtraggio

Il filtraggio è stato il passaggio più critico nella realizzazione dell'applicazione.

Supponiamo che l'utente stia cercando un Asus con 16 GB di RAM.

Cercando parola per parola all'interno dei prodotti di Amazon porterebbe ad avere come risultato dei prodotti che ad esempio hanno 16 GB di SSD, in quando ricerco 16 GB all'interno del testo del prodotto, e 8 GB di RAM. Questo perchè le parole 16 GB e RAM non sono correlate ma sono cercate in modo separato

all'interno del testo. Per ovviare a questo problema l'utente deve inserire le caratteristiche del prodotto che vuole cercare separate da una virgola. Sempre facendo riferimento al esempio in precedenza, l'utente dovrebbe inserire *Asus, 16GB RAM, ...* nel campo di ricerca. In questo modo è possibile distinguere le singole caratteristiche. Verrà quindi cercato nel testo *Asus* e poi *16 GB RAM*. Ciascuna caratteristica viene quindi cercata in blocco e non parola per parola.

Non basta però, perchè i prodotti di Amazon non hanno una rappresentazione comune delle caratteristiche, ma ognuno ha la propria. Ovvero è possibile che alcuni abbiano scritto *16GB RAM*, altri *GB 16 RAM* e cosi via. Si sono quindi resi necessari alcuni passaggi per manipolare le caratterisitche inserite dall'utente in modo da trovare il maggior numero di risultato possibili.

1. Nel testo del prodotto (quello che Amazon visualizza quando si ricerca un prodotto) vengono cercate tutte le **permutazioni** possibili della caratteristica. In riferimento all'esempio precedente vengono ricercate le seguenti parole 16 GB RAM, GB 16 RAM, ...

   Dopo di che si provano tutte le **combinazioni** togliendo alcuni spazi tra le parole. Ad esempio verrà cercato *16 GBRAM*, *16GB RAM*, ...

   Se un prodotto contiene tutte le caratteristiche direttamente nel testo del prodotto allora l'affidabilità di esso è al 100%.

2. Le caratteristiche che non sono state trovate vengono ricercate nella pagina del prodotto ovvero andando a consultare anche i dettagli di esso sempre utilizzando il meccanismo spiegato al punto 1.

   Se vengono trovate tutte le caratteristiche rimanenti allora l'affidabilità del prodotto è al 10%

3. Per le caratteristiche che ancora non sono state trovate si fa ancora un'ultima ricerca. Si cercano le parole della caratteristica senza che siano collegate tra di loro, non quindi come un blocco unico. Quindi verrà cercato 16 poi GB ed infine RAM. Il prodotto per essere restituito all'utente deve comunque contenere tutte le parole caratteristiche che l'utente ha inserito ma naturalmente l'affidabilità sarà inferiore. In questo caso l'affidabilità viene calcolata con queste formule:

$$W = \frac{(90 \times Tw)}{Cw} \tag{5.1}$$

$$L = \frac{(90 \times Tl)}{Cl} \tag{5.2}$$

$$R = 100 - \frac{(W + L)}{2} \tag{5.3}$$

   dove:

   - Tw è il numero totale di parole nella ricerca dell'utente

- Cw è il numero di parole che non sono state trovate con la ricerca precedente
- Tl è il numero totale di lettere nella ricerca dell'utente
- Cl è il numero di lettere delle parole che non sono state trovate con la ricerca precedente
- W è la percentuale di parole che non sono state trovate con la ricerca precedente
- L è la percentuale di lettere che non sono state trovate con la ricerca precedente
- R è l'affidabilità

Queste formule sono nate dal fatto che più parole non sono state trovate in precedenza, più significa che il prodotto potrebbe non essere conforme ma soprattutto più le parole sono grandi, in lettere, più queste sono importanti e quindi l'affidabilità deve essere inferiore.

Una volta completati tutti questi passaggi per ciascun prodotto risultante dalla ricerca su Amazon, vengono spediti all'applicazione solo quelli conformi.

Sull'applicazione, accanto a ciascun prodotto, è presente anche un bottone che permette di analizzare le recensioni del prodotto e quindi valutarlo in base a quelle.

5.3 Sentiment Analysis

Sentiment analysis è una tecnica che permette di elaborare il linguaggio naturale e di estrarre, identificare e studiare stati affettivi e informazioni soggettive. In particolare, quello che serve nella mia applicazione è determinare se una recensione (quindi del testo) è positivo o negativo. Questa tecnica viene spesso utilizzata nell'ambito del *Market research* e *Social media monitoring*.

Sentiment analysis è una sottocategoria dell'**NLP** ovvero una tecnica che misura l'inclinazione dell'opinione delle persone (Positiva, Negativa, Neutrale). Gli approcci che possono essere applicati sono due:

- **Rule-based**: è una tecnica che permette di analizzare del testo senza *training* e senza modelli di *Machine learning*. Utilizza un insieme di regole chiamate **Lexicons**.
- **Machine Learning based**

Come primo approccio ho scelto quello Rule-based utilizzando la libreria **NLTK**.

5.3.1 NLTK

"Natural Language ToolKit" è una libreria Python che contiene varie funzioni per l'analisi simbolica e statistica nel campo dell'elaborazione del linguaggio naturale. Permette di trasformare il linguaggio umano in modo che possa essere usato dai programmi. E' utilizzata per entrambi gli approcci definiti in precedenza.

5.4 Rule-based

Prima di tutto, tramite il web scraping, vengono estrapolate dall'HTML le recensioni del prodotto. Vengono estratte almeno 50 recensioni, ordinate dalla più recente, in modo tale da classificare il prodotto in modo migliore. Una volta estratte, per prima cosa è necessario prepararle e ripulirle.

5.4.1 Ripulitura

In questa fase vengono rimosse dalle recensioni i caratteri speciali ed i numeri dal testo in quanto ininfluenti quando si vanno ad analizzare. Ad esempio "Great CD: My lovely Pat has one of the GREAT voices of her generation." diventa "Great CD My lovely Pat has one of the GREAT voices of her generation". Questa operazione viene fatta con la semplicissima *Regular Expression* `[^A-Za-z]+`

Il testo però in questo modo non è analizzabile, quindi si passa alla fase di **Tokenization**

5.4.2 Tokenization

E' il processo che permette di spezzare la recensione i pezzi più piccoli chiamati **Token**. Questo permette di lavorare con parti più piccole che sono comunque ricche di significato anche se considerate separatamente rispetto al testo in cui sono. Esistono due tipi di Tokenization:

- **Tokenizing by word**: permette di identificare delle parole che compaiono con frequenza nel testo.
- **Tokenizing by sentence**: permette di capire come le parole sono relazione le une alle altre.

La tecnica che ho utilizzato è quella per parole. Grazie alla libreria NLTK è possibile utilizzare la funzione `nltk.word_tokenize("...")` che restituisce la lista di Token. Ad esempio "Great CD My lovely Pat has one of the GREAT voices of her generation" diventa

```
['Great', 'CD', 'My', 'lovely', 'Pat, 'has', 'one', 'of', 'the',...]
```

A questo punto si passa alla fase di **Stopwords removal**

5.4.3 Eliminazione delle "stopwords"

Le Stopwords sono delle parole che portano con sé pochissima informazione sul contenuto della frase, quindi, per semplicità possono essere rimosse. Per farlo si usa la funzione

```
("").join(e for e in tokens if e.lower() not in stopwords.words('english'))
```

Il risultato applicato al risultato precedente è

```
['Great', 'CD', 'lovely', 'Pat', 'one', ...]
```

Una volta fatto ciò è necessario effettuare la **Enrichment - POS tagging**.

5.4.4 Enrichment - POS tagging

Anche detta "Parts Of Speech tagging" è un processo che permette di trasformare i Token creati in precedenza in una lista avente la seguente forma `(parola, tag)`. Assegna un'etichetta alla parola in base al posto della frase in cui si trova. In inglese ci sono otto parti in una frase:

Parte	**Significato**	**Esempio**
Nome	Persona, luogo o cosa	mountain
Pronome	Sostituisce un nome	you, we
Aggettivo	Da un informazione rispetto ad un nome	efficient
Verbo	E' un azione o uno stato d'essere	learn, is
Avverbio	Da delle informazioni rispetto ad un verbo	efficently
Preposizione	Da delle informazioni rispetto a come un nome o un pronome sono collegati con un'altra parola	from
Congiunzione	Collega due parole o frasi	so
Interiezione	E' un esclamazione	wow

Con NLTK è possibile utilizzare la funzione `ntlk.pos_tak(token)` che restituisce un lista con ad ogni parola il tag associato. Ad esempio

```
['Great', 'CD', 'My', 'lovely', 'Pat, 'has', 'one', 'of', 'the',...]
```

diventa

```
[('Great', 'NNP'), ('CD', 'NN'), ('lovely', 'RB'),...]
```

5.4.5 Ottenimento delle "steam words"

Lo **steam** è una parte del testo che è responsabile del significato lessicale della frase. Ci sono due tecniche che possono essere utilizzate per estrapolare dalle parole le *steam words*:

- **Stemming**: è una tecnica meno accurata perchè si occupa solo di troncare il finale delle parole e spesso restituisce parole senza significato
- **Lemmatization**: è una tecnica che restituisce solo parole aventi significato. Necessità però che ciascuna parola abbia un **POS tag**.

Ho scelto di utilizzare la tecnica della **Lemmatization** in quanto risulta più precisa e affidabile. Grazie alla funzione `wordnet_lemmatizer.lemmatize(word, pos=pos)` e possibile ottenere la *steam word* della parola passata come parametro. Una volta fatto questo le parole vengono ricomposte per formare una frase. Ad esempio

```
[('Great', 'NNP'), ('CD', 'NN'), ('lovely', 'RB'),...]
```

diventa "Great CD love ...".

5.4.6 VADER

Valance Aware Dictionary and sEntiment Reasoner è un tool Rule-based che permette di effettuare il *Sentiment Analysis*. **VADER** usa un dizionario in cui ogni *lexicon* è associato ad un punteggio in base all'intensità emotiva e all'orientamento semantico(positività o negatività). Questa tecnica non restituisce solo la positività o la negatività di una frase ma anche quanto una frase è positiva o negativa.

Il punteggio di ciascuna frase è ottenuto andando a sommare il punteggio di ciasuna parola nel testo.

Il vantaggio più importante che offre questa tecnica è che non richiede alcun *training data* quindi può essere utilizzato subito.

5.5 Machine Learning based

E' una tecnica, alternativa alla precedente, che permette di costruire dei **modelli**, basati su dei dati semplici, chiamati **training data**, in modo da fare predizioni o decisioni senza essere esplicitamente programmati per farlo.

Un **modello** premette di descrivere un sistema utilizzando regole ed equazioni.

5.5.1 Classificatore Bayesiano

Questo tipo di classificatore si basa sull'algoritmo **Naive Bayes** che utilizza il teorema di **Bayes**. Il teorema di Bayes permette di calcolare la probabilità condizionata di un evento A rispetto ad un altro evento B sapendo che B si è verificato.

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{5.4}$$

L'obbiettivo è quello di capire, date una serie di recensioni pre-classificate, se una nuova recensione è positiva o negativa.

Una caratteristica molto importante di questo algoritmo la **bag-of-words**. Questo implica che all'algoritmo importa solo la frequenza delle parole ma non la posizione che queste parole hanno.

L'approccio utilizzato dall'algoritmo è questo:

1. Supponiamo di avere un insieme di possibili classi da assegnare C = Positiva, Negativa.

2. Dobbiamo calcolare la probabilità che una recensione appartenga ad una delle classi.

3. Ovvero dobbiamo eseguire un'iterazione su tutte le classi possibili e trovare quella che ha la probabilità condizionata più alta.

I passi successimi mostrano com'è possibile derivare la formula utilizzata dall'algoritmo

$$c* = argmax\ P(c|d)\ c \in C \tag{5.5}$$

dove c* indica la classe che ha la massima probabilità.

E' quindi possibile andare a sostituire P(c—d) utilizzando il teorema di Bayes

$$c* = argmax \frac{P(d|c) \times P(c)}{P(d)}\ c \in C \tag{5.6}$$

Possiamo semplificare questa formula: quando effettuiamo l'iterazione rispetto alle classi presenti in C, la recensione non cambia, cambiano solo le classi. Quindi possiamo rimuovere P(d)

$$c* = argmax\ P(d|c) \times P(c)\ c \in C \tag{5.7}$$

A questo punto possiamo considerare la recensione come un insieme di parole quindi la formula diventa

$$c* = argmax\ P(w_1, w_2, ..., w_n|c)P(c)\ c \in C \tag{5.8}$$

Per continuare è necessario effettuare un'assunzione molto importante, ovvero dobbiamo supporre che la probabilità di ciascuna w (parola) sia **indipendente** dalle altre. Possiamo quindi ulteriormente semplificare la formula

$$c* = argmax\ P(c) \times \prod P(w_i|c)\ c \in C,\ w_i = parola\ i - esima \tag{5.9}$$

Una volta derivata la formula possiamo applicarla. Nei passi successivi viene mostrato un esempio di com'è possibile utilizzare la formula.

1. Calcoliamo P(c). Dobbiamo quindi trovare il numero di recensioni che appartengono a ciascuna classe. Supponiamo che il numero di recensioni di classe c siano *Nc* e il numero totale sia *Ntotal*.

$$P(c) = \frac{Nc}{Ntotal} \tag{5.10}$$

2. Calcoliamo $P(w_i|c)$. Il nostro obbiettivo è quello di calcolare il numero di volte che la parola w_i compare nel *training dataset* relativo alla classe c.

$$P(w_i|c) = \frac{count(w_i, c)}{\sum_{w_i \in V} count(w_i, c)} \tag{5.11}$$

dove V è l'insieme di tutte le parole presenti nel dataset considerato. Esiste però un problema: in alcuni casi, il valore calcolato da questa formula può essere 0. Per evitare ciò si introduce un coefficiente chiamato **Laplace Smoothing Coefficient**(a) sia al numeratore che al denominatore.

$$P(w_i|c) = \frac{count(w_i, c) + a}{\sum_{w_i \in V}(count(w_i, c) + a)} \tag{5.12}$$

3. Si introducono i risultati calcolati precedentemente (1. e 2.) nella formula 5.9.

Grazie alla formula generata al passo 3. è quindi possibile calcolare se una recensione è positiva o negativa.

Una volta capiti i meccanismi che stanno alla base del classificatore Bayesiano è stato possibile passare alla sua implementazione.

5.5.2 Implementazione

Per poter adottare questo tipo di approccio si è reso necessario un **dataset** di partenza, ovvero una collezione di recensioni già classificate, ovvero recensioni che hanno un'etichetta che indica se sono positive o negative. Il dataset però deve essere manipolato per poter essere utilizzato, si procede quindi con i passaggi descritti in precedenza (5.4.1 - 5.4.5). Questi passaggi permettono di mantenere nel dataset ciò che effettivamente serve all'interno di una recensione per poterla classificare.

A questo punto si passa alla fase di **allenamento (training)**. E' un passo molto importante perchè permette al classificatore di interpretare correttamente i dati ed imparare da essi in modo da eseguire le future classificazioni nel modo più corretto possibile. Infatti fornendo dati di allenamento, il classificatore cercherà di predire la classe corretta di quella determinata recensione. Man mano, grazie a degli aggiustamenti, si avvicinerà sempre di più alla classificazione corretta.

Tramite l'istruzione

```
nltk.NaiveBayesClassifier.train(train_set)
```

il classificatore viene allenato.

Dopo di che, tramite l'istruzione

```
nltk.classify.accuracy(classifier, test_set)
```

è stato possibile valutare l'accuratezza del classificatore. Questa funzione confronta le classi predette dal classificatore rispetto alle classi preimpostate nel dataset. L'accuratezza del classificatore creato è risultata al **71%**, che è un ottimo risultato. Naturalmente può essere migliorato ampliando il *dataset*.

Il classificatore allenato deve poi essere salvato in modo tale che possa essere utilizzato per predire le classi delle recensioni quando l'utente lo richiederà.

Capitolo 6

Ottimizzazione

Dopo una serie di test di ricerca sull'applicazione, come anche spiegato in precedenza, è stato stimato che per analizzare 25 prodotti ci vogliono circa 3 minuti. Questo ha portato quindi a cercare delle soluzioni che potessero ottimizzare la ricerca in modo da restituire dei risultati all'utente nel minor tempo possibile.

6.0.1 Parallelizzazione ricerca

Per prima si è reso necessario parallelizzare la ricerca ciò cercare le caratteristiche richieste su più prodotti in modo contemporaneo. Quindi si è pensato di utilizzare un **Thread pool** di lunghezza varabile (in base alla disponibilità di risorse hardware). Ad ogni thread viene passata l'istanza del browser su cui eventualmente aprire la pagina del prodotto ed il prodotto da analizzare. Il thread, se non trova tutte le caratteristiche richieste dall'utente nel nome del prodotto, analizza l'intera pagina del prodotto. Nello specifico, ciascun thread, è un **Future task** ovvero un oggetto che ritornerà un valore nel futuro, infatti ritornerà il prodotto se esso è conforme alle richieste oppure un JSON vuoto nel caso contrario. Per quanto riguarda invece l'utilizzo del Tread pool, ho deciso di utilizzare quello, invece che creare thread all'occorrenza, in modo tale da risparmiare ancora più tempo perchè i thread vengono creati una sola volta all'inizio e vengono riutilizzati "sprecando" cosi meno tempo. Applicando le ottimizzazioni descritte sopra sono riuscito a portare il tempo per analizzare 25 prodotti a circa 1/2 minuti. Era però ancora troppo.

6.0.2 Fast trasmit

Il tempo impiegato per restituire i risultati all'utente era però ancora troppo. Ho deciso così di apportare ancora un'ottimizzazione, ovvero ogni 5 prodotti conformi con le caratteristiche richieste dall'utente, essi vengo subito inviati come risultato all'applicazione. In questo modo l'utente, dopo circa 30 secondi (stima), può già consultare dei prodotti. Nel frattempo, in background tramite altre richieste fatte dall'applicazione alla stessa risorsa server, vengono analizzati i prodotti rimanenti sempre utilizzando questo tipo di ottimizzazione.

Sono quindi passato dal restituire i prodotti in 3 minuti a restituire alcuni prodotti dopo solo 30 secondi. Naturalmente per le tempistiche della rete resta comunque alto ma bisogna ricordarsi che l'utente non dovrà perdere ulteriore tempo perchè i risultati sono solo quelli conformi con quello che lui ha richiesto.

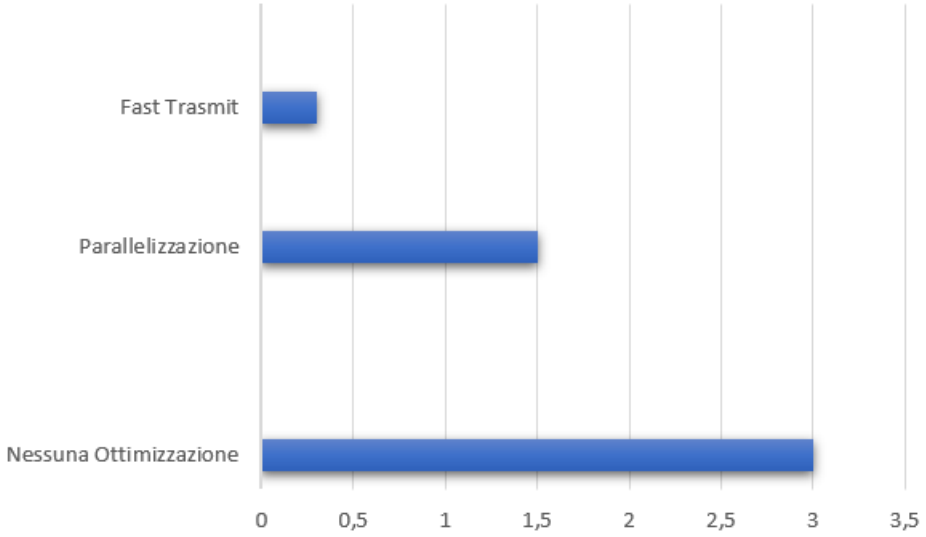

Figura 6.1: Grafico delle ottimizzazioni

Capitolo 7

Conclusioni

7.1 Difficoltà riscontrate

In questa sezione saranno esposte le maggiori problematiche riscontrate durante lo sviluppo dell'applicazione.

7.1.1 Problema del filtraggio

Il filtraggio è il punto cardine del progetto sviluppato. E' stato studiato nel dettaglio per garantirne la massima efficienza ma soprattutto per migliorare quella che è l'esperienza utente.

Durante la fase di studio, la difficoltà più importante che è stata riscontrata è quella di capire come cercare nei prodotti di un E-commerce le caratteristiche richieste dall'utente. Non è possibile cercare l'intera stringa inserita dall'utente nella barra di ricerca in quanto gli annunci hanno informazioni strutturate, mentre la stringa no. La soluzione a questo problema è possibile trovala al paragrafo 5.2.4. La ricerca della suddetta ha comportato ad un'analisi dettagliata riguardo alla composizione testuale degli annunci e alle possibili ricerche che un utente può fare. Questo ha comportato un consumo quantitativo del tempo da me dedicato.

7.2 Limitazioni

Una grande limitazione che è stata riscontrata deriva dall'utilizzo di **Selenium** seppur sia molto più veloce della tecnica utilizzata all'inizio ovvero **Beautiful Soup**.

Siccome, per poter navigare, usa delle finestre di un browser, la velocità complessiva dello scraping diminuisce molto. Infatti, parte del tempo viene persa solamente per aprire una nuova finestra del browser, mentre lo scraping del contenuto della pagina viene fatto in modo quasi istantaneo. Tutte le volte che bisogna aprire un link, cliccare un pulsante o simili, si perde molto tempo. Questa perdita di tempo deve essere sommata alla velocità della connessione perchè più questa è lenta più la velocità di risposta del browser diminuisce.

Quindi, la velocità dell'applicazione, dipende molto dalla potenza del calcolatore su cui è installato il server e la velocità della rete a cui è collegato.

7.3 Sviluppi futuri

7.3.1 Miglioramento filtraggio

Per prima cosa si potrebbe migliorare il modo in cui vengono cercate le caratteristiche del prodotto. Si dovrebbe, per ogni parola, trovare tutti i possibili sinonimi che i siti di E-commerce utilizzano e cercare anche quelli, oltre alle parole originali inserite dall'utente.

Infatti spesso, durante la prima fase di analisi delle ricerche, è stato riscontrato questo problema: cercando ad esempio "Asus, 16 GB RAM" gli E-commerce producono anche dei risultati in cui "RAM" è definita con il termine "DDR4" che è un tipo specifico di "RAM".Si potrebbero, quindi, inserire dei meccanismi di **Machine learning** che riescano ad imparare quali sono i sinonimi più utilizzati dagli annunci rispetto ad una parola cercata dall'utente.

7.3.2 Multiutenza

Un altro sviluppo sarebbe quello di inserire la possibilità di poter far usufruire l'applicazione da più utenti. Ora l'applicazione è pensata per un solo utente in quanto l'obbiettivo del tirocinio era sullo scraping e sul filtraggio, non sulla multi-utenza già ampiamente analizzata durante tutto il mio corso di studi.

7.4 Applicazioni simili

Purtroppo, o per fortuna, non esistono applicazioni simili che effettuano queste operazioni in quanto, viene dato per scontato, che i siti di E-commerce, permettano di effettuare il filtraggio dei risultati. In effetti è così, infatti è possibile scegliere, durante una ricerca, le caratteristiche che un prodotto dovrà avere. Però, quello che i siti di E-commerce fanno, è mescolare i risultati conformi alle caratteristiche con quelli che devono essere venduti secondo le politiche aziendali.

Questa applicazione è stata creata per mettere alla luce questi problemi e cercare di risolverli, ma soprattutto per spingere gli utenti di questi E-commerce a non valutare l'acquisto solo dei primi prodotti che risultano da una ricerca ma approfondirla anche confrontando più siti web in modo tale da non cadere nella "trappola" delle multinazionali.

7.5 Link GitHub

Durante tutto lo sviluppo dell'applicazione ho utilizzato GitHub come supporto per il versioning del mio codice. E' possibile reperire il risultato della suddetta

tesi al seguente link `https://github.com/fabriziosanino/bachelor-degree.git`

Bibliografia

- **Instant Developer**, *Filtri nelle app e nei siti: ecco perché creare quelli giusti è difficile, ma indispensabile*, https://www.instantdeveloper.com/blog/ux-ui/filtri-nelle-app-e-nei-siti-ecco-perche-creare\-quelli-giusti-e-difficile-ma-indispensabile/

- **Real Python**, *Python and MySQL Database: A Practical Introduction*, https://realpython.com/python-mysql/

- **Real Python**, *Sentiment Analysis: First Steps With Python's NLTK Library*, https://realpython.com/python-nltk-sentiment-analysis/

- **Analytics Vidhya**, *Rule-Based Sentiment Analysis in Python*, https://www.analyticsvidhya.com/blog/2021/06/rule-based-sentiment-analy\-sis-in-python/

- **Digital Ocean**, *How To Perform Sentiment Analysis in Python 3 Using the Natural Language Toolkit (NLTK)*, https://www.digitalocean.com/community/tutorials/how-to-perform-sentiment-analysis-in-python\-3-using-the-natural-language-toolkit-nltkstep-6-preparing-data\-for-the-model

- **Full Stack Python**, *Flask Web developement*, https://www.fullstackpython.com/flask.html

- **Real Python**, *Beautiful Soup: Build a Web Scraper With Python*, https://realpython.com/beautiful-soup-web-scraper-python/

- **Analytics Vidhya**, *Performing Sentiment Analysis With Naive Bayes Classifier!*, https://www.analyticsvidhya.com/blog/2021/07/performing\-sentiment-analysis-with-naive-bayes-classifier/

- **Geeksforgeeks**, *Abstract Classes in Python*, https://www.geeksforgeeks.org/abstract-classes-in-python/

- **Towards Data Science**, *SENTIMENTAL ANALYSIS USING VADER*, https://towardsdatascience.com/sentimental-analysis-using-vader\-a3415fef7664

- **Towards Data Science**, *A Hitchhiker's Guide to Sentiment Analysis using Naive-Bayes Classifier*, https://towardsdatascience.com/a-hitchhikers\-guide-to-sentiment-analysis-using-naive-bayes-classifier-b921c0fb694

Ringraziamenti