Università degli Studi di Napoli Parthenope Corso: Programmazione 3

Marketing comportamentale

Andrea Altobelli

Friday 25th June, 2021

Indice

1 Traccia

Si vuole simulare un sistema per la vendita di prodotti basato su marketing comportamentale. Ogni prodotto è identificato da un codice, da un nome, una descrizione, una quantità di scorta, il costo e la categoria. Scrivere un programma per la vendita dei prodotti. L'accesso può essere effettuato in modalità amministratore e in modalità utente. L'amministratore può effettuare le seguenti operazioni:

- inserire o cambiare un nuovo prodotto con le sue informazioni
- in base agli acquisti effettuati dagli utenti, proporre offerte specifiche
- raggruppare gli utenti in base agli acquisti effettuati

L'utente può effettuare le seguenti operazioni:

- inserire i prodotti nel carrello della spesa
- effettuare il pagamento. Il pagamento può avvenire secondo le modalità: contanti, carta di credito o bancomat.

1.1 Marketing comportamentale

Il marketing comportamentale si prefigge di mostrare annunci pubblicitari per singolo utente in base all'analisi del comportamento dello stesso. Si suppone di usare una versione semplificata della metodologia bag of words con tecnica term frequency inverse document frequency (TF-IDF). Per ogni categoria sono presenti un numero di prodotti. Per le offerte specifiche l'amministratore controlla, per il singolo utente, la categoria che presenta, globalmente, maggiore acquisti.

1.2 Clustering

Il clustering o analisi dei gruppi è un insieme di tecniche di analisi dei dati volta alla selezione e raggruppamento di elementi omogenei in un insieme di dati. In questo contesto usare l'algoritmo K-Means sui vettori identificati da tutti i prodotti, come nella fase di Marketing comportamentale, e per tutti gli utenti.

2 Framework

2.1 Spring MVC

Il progetto consiste in un'applicazione Web realizzata con framework Spring su piattaforms Java. In particolare l'architettura della Web application è realizzata con Spring MVC, che sfrutta il pattern MVC costitutito da 3 componenti:

- **Model** che si occupa di accedere ai dati necessari alla logica di business ossia oggetti di gestione o classi di accesso al database
- **View** che si occupa di creare l'interfaccia utilizzabile dall'utente e che espone i dati da esso richiesti. Per questa componente è utilizzato Thymeleaf, ossia un motore di template Java XML / XHTML / HTML5. Nel caso specifico dell'applicazione i template sono scritti in HTML 5, con Javascript, jQuery, e framework CSS Bulma.
- **Controller** implementano la logica di controllo, riceve i comandi dell'utente (attraverso lo strato View) e li attua modificando lo stato degli altri due componenti. Nello specifico dell'applicazione i Controller sono rappresentati da classi (chiamate appositamante @Controller) che rimangono "in ascolto" su un determinato URL e, grazie ai Model e alle View, si occupano di gestire la richiesta dell'utente. Anche se i Controller sono molteplici, le richieste vengono inviate comunque ad unico Handler Mapping che seleziona il Controller adeguato.

La web application viene eseguita su un web container Apache Tomcat 7, istanziato da Spring di default.

2.2 Spring Security

Spring Security è un framework di Spring utilizzato nella Web application per l'autenticazione e le autorizzazioni degli utenti. Queste funzioni sono implementate nella classe **WebSecurityConfig**.

Nello specifico l'interfaccia di login che viene sfruttata è quella di base di Spring Security, che è configurata attraverso la funzione configure, la quale fornisce all'AuthenticationManagerBuilder i dettagli dell'utente attraverso la classe UserDetailsServiceImp. Questi dettagli sono caricati direttamente dal database attraverso il DAO e la sua funzione findUsername() che li restitusice in base all'username.

Per quanto riguarda le autorizzazioni, sempre nella classe **WebSecurityConfig**, un'altra funzione configure(), gestisce le richieste in base al ruolo degli utenti che può essere **ADMIN** o **CLIENTE**.

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    };

    public UserDetailsService userDetailsService() {
        return new UserDetailsServiceImp();
    };

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService()).passwordEncoder(passwordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .antMatchers("/home")
                .permitAll()
                .antMatchers("/")
                .permitAll()
                .antMatchers("/register")
                .permitAll()
                .antMatchers("/registerProcess")
                .permitAll()
                .antMatchers("//img/shopping-online.jpg")
                .permitAll()
                .antMatchers("/img/logo.jpg")
                .permitAll()
                .antMatchers("/adminPanel")
                .hasRole("ADMIN")
                .antMatchers("/groupBy/**")//con gli  specifico tutti gli indirizzi a partire da /groupBy/
                .hasRole("ADMIN")
                .antMatchers("/sendOffer")
                .hasRole("ADMIN")
```

3 Database

Il database utilizzato è MariaDB che è un database relazionale. La connessione al databsae avviene con la classe Connect attraverso l'URL.

```
public class Connect {

    private Connection connection = null;

    private static final String DATABASE_DRIVER = "org.mariadb.jdbc.Driver";
    private static final String DATABASE_URL = "jdbc:mariadb://localhost:3306/test?user=root&password=root";

    private String jdbcDriver;
    private String databaseUrl;

    Connect() {
        Properties properties = new Properties();
        try {
            jdbcDriver = properties.getProperty("driver", DATABASE_DRIVER);
            databaseUrl = properties.getProperty("url", DATABASE_URL);
            connection = DriverManager.getConnection(DATABASE_URL);
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }

    public Connection getConnection() {
       return connection;
    }

}
```

4 Pattern

I pattern utilizzati nella realizzazione della Web application sono:

- **Singleton** per l'implementazione del DAO. In questo modo è garantito un unico punto di accesso per richiedere operazioni al database
- **State** per l'implementazione dell'utente. Il comportamento degli utenti in relazione a delle operazioni, è in questo modo regolato in base alla tipologia di User (ADMIN o CLIENTE).
- **Visitor** per l'interazione con i prodotti. Siccome le varie tipologie di prodotti presentano caratteristiche diverse, le istruzioni da eseguire per la stessa operazione cambiano a seconda della tipologia. Il visitor è in grado di selezionare la funzione opportuna.
- **Strategy** per le diverse metodologie di pagamento. Questa implementazione consente di modificare dinamicamente le istruzioni in base al tipo di pagamento scelto dall'utente.
- **Prototype** per ottenere diversi tipi di prodotti non conoscendo in partenza la loro specifica classe. Infatti nella funzione getProduct() del DAO, nel momento che viene effettuata una query per id non so che tipo di prodotto mi venga restituito. La classe dell'oggetto è definita a runtime, prima che si avvii la funzione si è solo certi di avere un oggetto dell'interfaccia comune a tutti prodotti. Non conoscendo quindi la classe non posso istanziare un nuovo prodotto, e, con una query successiva andrei a sovrascrivere il prodotto ottenuto in precedenza. Perciò è conveniente clonare l'ultimo oggetto sfruttando il metodo clone comune a tutti i prodotti, ottenendo quindi un istanza separata che ha uno spazio di indirizzamento diverso.

5 Marketing comportamentale

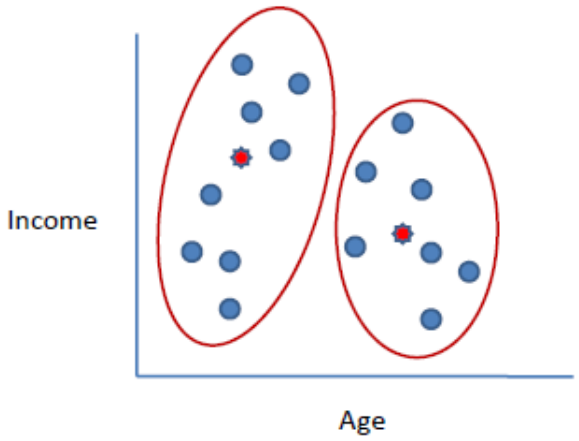
Lo scopo è di proporre offerte ai singoli utenti in base al comportamento degli stessi. L'analisi si compone di diversi passaggi.

5.1 Tabella delle abitudini di acquisto

Innanzitutto viene creata una tabella che rappresenta le abitudini di acquisto dei clienti registrati sulla piattaforma. Questa è una versione semplificata del TF-IDF, ed è costituita da una matrice di interi dove, l'indice di riga rappresenta l'utente, l'indice di colonna una categoria di prodotti, ed infine i valori all'interno saranno il numero di acquisti dell'utente per categoria. In questo modo sarà possibile esaminare le abitudini di acquisto di un utente esaminando la corrispondente riga all'interno della matrice.

5.2 Clustering

Ottenuta la matrice, l'algoritmo Kmeans prende in considerazione gli utenti e le loro abitudini di acquisto. Questi utltimi sono trasposti in uno spazio n-dimensionale (numero di categorie di prodotti) come punti, le cui coordinate sono le categorie di prodotti, e i valori delle coordinate ottenute dai corrispondenti valori della matrice che descrivono il numero di accquisti dell'utente per categoria. A questo punto inizia l'algoritmo che assegna i punti in uno dei k cluster differenti, scegliendo all'inizio centroidi (centri dei cluster) randomicamente. Il numero di cluster k è inserito in input dall'utente. L'algoritmo ripete iterativamente la scelta dei centroidi e l'assegnazione dei punti ai cluster fino a che lo scostamento delle distanze dei centroidi non scende sotto una determinata soglia. Alla fine avremo gli utenti raggruppati in k cluster.



L'esempio sopra mostra un caso semplificato con 2 dimensioni, Income e Age.

5.3 Calcolo dell'istogramma

Completata la fase di clustering, attraverso l'interfaccia utente si accede alla schermata dedicata ad un cluster tra i k disponibili. Viene così calcolato l' istogramma dei prodotti più acquistati dagli utenti del cluster. Come si evince dalla figura, il sistema propone all'Admin di inviare ad un prezzo scontato l'oggetto con maggiori acquisti.