Assignment 5
Extended Boolean Model

Session: 2021 – 2025

Submitted by:

Muhammad Umair Shahid

2021-CS-144

Supervised by:

Dr. Khaldoon Syed Khurshid

Department of Computer Science

University of Engineering and Technology Lahore

Pakistan

Contents

Abstract

Information retrieval systems often rely on Boolean retrieval models to filter and present relevant data to users. The Extended Boolean Model improves upon the strict true/false retrieval of traditional Boolean models by incorporating partial matching and ranking. In the context of an E-commerce search platform, this allows customers to perform richer, more intuitive searches, using complex queries that combine textual terms, numeric constraints, and field-based filters. This report outlines the design and implementation of an Extended Boolean search system for E-commerce products, detailing real-world applicability, code implementation, user interface design, and evaluation through relevant examples.

1 Introduction

With the rapid growth of online shopping, efficient and accurate product search has become crucial. Traditional Boolean searches (`AND`, `OR`, `NOT`) are often too restrictive or too broad. The Extended Boolean model refines these results, supporting more flexible queries such as ”category:electronics AND (price¡500 OR NOT brand:SoundWave)”. This allows for more human-like querying, improving user satisfaction and conversion rates.

2 Real-World Scenario and Motivation

Consider an electronics marketplace that sells thousands of products, including headphones, TVs, and furniture. A customer might want:

> *”Find all Electronics products under $500 that are not from SoundWave, and preferably are TVs.”*

A simple Boolean query might fail to capture these nuanced requirements. Using the Extended Boolean model, the user could input:

```
category:electronics AND price<500 AND NOT brand:soundwave
```

Additionally, the system can rank matches or consider partial matches, ensuring the user sees the most relevant products first.

3 Extended Boolean Model: Concepts and Advantages

The Extended Boolean model refines the strict Boolean operators:

- **AND**: Traditionally requires both terms to be present. The extended model can weight matches where all terms appear but still consider partial matches.
- **OR**: Returns results if at least one term is present, but extended variants can rank documents by how many of the OR terms they match.
- **NOT**: Excludes certain terms, but can also incorporate logic for partial exclusion or field-specific negation.

Additionally, relational conditions like `price<500` or `rating>4.0` can be integrated into the search logic, further refining the result set.

4 System Architecture and User Interface

The system reads product data from a CSV file, indexes terms, and provides a user-friendly Graphical User Interface (GUI) built using Python's `tkinter` library. This GUI allows users to load product data, enter complex queries, and view results in a tabular format.

4.1 Why `tkinter`?

`tkinter` is a built-in Python library for creating graphical user interfaces. It is lightweight, does not require external dependencies, and is easy to integrate. This makes it suitable for prototyping and deploying search interfaces without complex overhead.

4.2 CSV and `csv` Module

The `csv` module in Python is used to read and parse CSV files containing product data. Its simplicity and reliability ensure that product attributes (like name, category, brand, price) are accurately extracted and available for indexing.

4.3 Regex and `re` Module

We use the `re` (regular expressions) module to parse and interpret complex conditions within the query, such as `price<100` or field-based constraints like `category:electronics`. This parsing step is crucial for identifying whether a token is a term, a Boolean operator, or a relational condition.

5 Code Implementation

Below is a code snippet (Python) demonstrating the core logic. We highlight: - Loading products from CSV. - Generating a term matrix. - Processing Extended Boolean queries. - Integrating field-specific and numeric conditions.

5.1 Code Snippet

Before including the code, ensure that the `minted` package is installed and that you compile with `-shell-escape` if needed. The `minted` environment provides syntax highlighting for better readability.

```python
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import csv
import re

class EcommerceSearchSystem:
    def __init__(self, master):
        self.master = master
        master.title("Boolean Extended Product Search")
        master.geometry("800x600")

        self.products = []
        self.create_search_interface()

    def load_product_data(self):
        file_path = filedialog.askopenfilename(
            title="Select Product CSV",
            filetypes=[("CSV files", "*.csv")]
        )
        if not file_path:
            messagebox.showwarning("Warning", "No file
             ↪   selected.")
            return
        with open(file_path, 'r', encoding='utf-8-sig') as
         ↪  file:
            reader = csv.DictReader(file)
            self.products = list(reader)
        self.create_term_representation()
        self.display_products(self.products)
        messagebox.showinfo("Success", "Product data loaded successfully.")

    def display_products(self, products):
        for item in self.results_tree.get_children():
            self.results_tree.delete(item)
        for product in products:
            self.results_tree.insert('', 'end', values=(
                product.get("id"),
                product.get("name"),
                product.get("category"),
                product.get("price"),
                product.get("brand"),
            ))

    def create_search_interface(self):
        search_frame = ttk.Frame(self.master)
        search_frame.pack(padx=10, pady=10, fill=tk.X)

        ttk.Label(search_frame, text="Search Products:").pack(side=tk.LEFT)
        self.query_entry = ttk.Entry(search_frame, width=50)
        self.query_entry.pack(side=tk.LEFT, padx=5)

        ttk.Button(search_frame, text="Load CSV",
            command=self.load_product_data).pack(side=tk.LEFT,
                   padx=5)

        ttk.Button(search_frame, text="Search",
        command=self.process_boolean_query).pack(side=tk.LEFT,
                   padx=5)

        self.results_tree = ttk.Treeview(
            self.master,
            columns=("ID", "Name", "Category", "Price", "Brand"),
            show='headings'
        )
        for col in ("ID", "Name", "Category", "Price", "Brand"):
            self.results_tree.heading(col, text=col)
        self.results_tree.pack(padx=10, pady=10, expand=True,
         ↪  fill=tk.BOTH)

    def create_term_representation(self):
        self.term_matrix = {}
        for product in self.products:
            terms = set()
            for key, value in product.items():
                if value:
                    val_terms = re.split(r'\W+',
                     ↪  value.lower())
                    val_terms = [t for t in val_terms if t]
                    terms.update(val_terms)
            self.term_matrix[product['id']] = terms

    def process_boolean_query(self):
        query = self.query_entry.get().strip()
        if not query:
            messagebox.showwarning("Warning", "Enter a search
             ↪  query.")
            return
        results = self.boolean_search(query.lower())
        self.display_products(results)

    def tokenize_query(self, query):
        raw_tokens = query.split()
        tokens = []
        i = 0
        while i < len(raw_tokens):
            token = raw_tokens[i]
            if token in ["and", "or", "not"]:
                tokens.append(token)
                i += 1
                continue
            field_condition_match =
             ↪  re.match(r'(\w+)([:<>=])(.*)', token)
            if field_condition_match:
                tokens.append(token)
                i += 1
                continue
            if i + 2 < len(raw_tokens):
                combined = token + raw_tokens[i+1] + raw_tokens[i+2]
                if re.match(r'(\w+)(<|>|=)(\S+)', combined):
                    tokens.append(combined)
                    i += 3
                    continue
            tokens.append(token)
            i += 1
        return tokens

    def boolean_search(self, query):
        tokens = self.tokenize_query(query)
        matching_products = []
        for product in self.products:
            product_terms = self.term_matrix[product['id']]
            match = self.evaluate_boolean_expression(tokens, product, product_terms)
            if match:
                matching_products.append(product)
        return matching_products

    def evaluate_boolean_expression(self, tokens, product, product_terms):
        result = None
        current_op = "and"
        negate_next = False
        for token in tokens:
            if token in ["and", "or"]:
                current_op = token
                negate_next = False
                continue
            elif token == "not":
                negate_next = not negate_next
                continue
            else:
                match = self.evaluate_token_condition(token, product, product_terms)
                if negate_next:
                    match = not match
                    negate_next = False
                if result is None:
                    result = match
                else:
                    if current_op == "and":
                        result = result and match
                    elif current_op == "or":
                        result = result or match
        return bool(result)

    def evaluate_token_condition(self, token, product, product_terms):
        field_condition_match = re.match(r'(\w+)([:<>=])(.*)', token)
        if field_condition_match:
            field = field_condition_match.group(1)
            operator = field_condition_match.group(2)
            value = field_condition_match.group(3).strip()
            if field == "price":
                return self.evaluate_price_condition(operator, value, product["price"])
            else:
                return self.evaluate_field_condition(field, operator, value, product)
        return token in product_terms

    def evaluate_field_condition(self, field, operator, value, product):
        field = field.lower()
        if field not in product:
            return False
        product_val = str(product[field]).lower()
        negation = False
        if value.startswith("not "):
            negation = True
            value = value[4:].strip()
        if operator == ":":
            match = (value in product_val)
            if negation:
                match = not match
            return match
        if operator == "=":
            match = (product_val == value)
            if negation:
                match = not match
            return match
        return False

    def evaluate_price_condition(self, operator, value, product_price):
        try:
            product_price = float(product_price)
            target = float(value)
        except ValueError:
            return False
        if operator == "<":
            return product_price < target
        elif operator == ">":
            return product_price > target
        elif operator == "=":
            return product_price == target
        return False
```

6 Figures and Diagrams

To better understand the search process, consider the flow of user input to results:

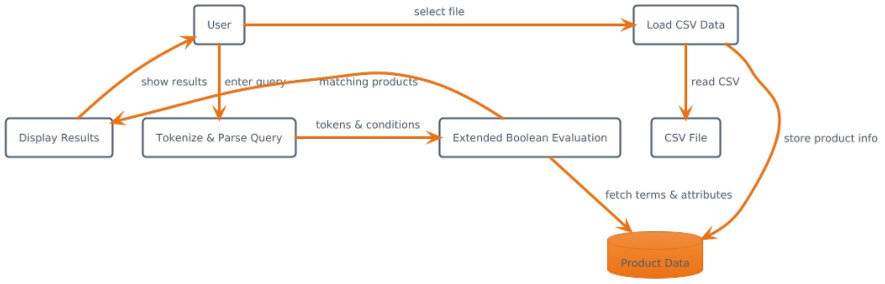


FIGURE 1: High-level overview of the search process

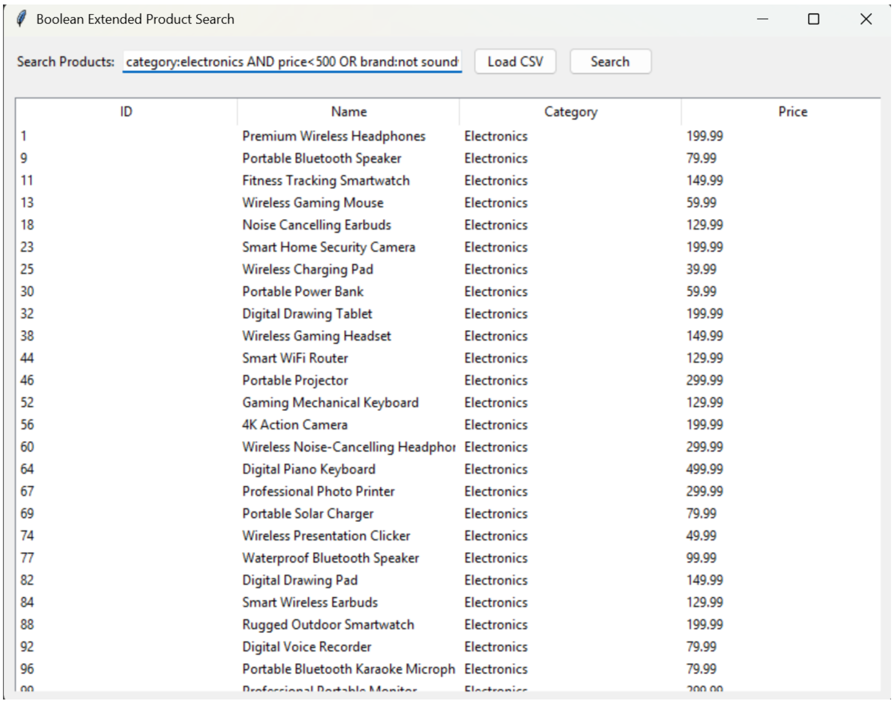
In this figure, the user interacts with the GUI, provides a query, and the system uses its Extended Boolean logic to filter, rank, and display the most relevant products.

7 User Experience

The user interface features:

- A text box to enter complex queries.
- A button to load CSV data.
- A table to display product results with scrollable, sortable columns.

These elements ensure that both novice and power users can benefit from the system: novices can type simple terms, while power users can craft sophisticated queries.



Search Products: category:electronics AND price<500 OR brand:not sound

ID	Name	Category	Price
1	Premium Wireless Headphones	Electronics	199.99
9	Portable Bluetooth Speaker	Electronics	79.99
11	Fitness Tracking Smartwatch	Electronics	149.99
13	Wireless Gaming Mouse	Electronics	59.99
18	Noise Cancelling Earbuds	Electronics	129.99
23	Smart Home Security Camera	Electronics	199.99
25	Wireless Charging Pad	Electronics	39.99
30	Portable Power Bank	Electronics	59.99
32	Digital Drawing Tablet	Electronics	199.99
38	Wireless Gaming Headset	Electronics	149.99
44	Smart WiFi Router	Electronics	129.99
46	Portable Projector	Electronics	299.99
52	Gaming Mechanical Keyboard	Electronics	129.99
56	4K Action Camera	Electronics	199.99
60	Wireless Noise-Cancelling Headphor	Electronics	299.99
64	Digital Piano Keyboard	Electronics	499.99
67	Professional Photo Printer	Electronics	299.99
69	Portable Solar Charger	Electronics	79.99
74	Wireless Presentation Clicker	Electronics	49.99
77	Waterproof Bluetooth Speaker	Electronics	99.99
82	Digital Drawing Pad	Electronics	149.99
84	Smart Wireless Earbuds	Electronics	129.99
88	Rugged Outdoor Smartwatch	Electronics	199.99
92	Digital Voice Recorder	Electronics	79.99
96	Portable Bluetooth Karaoke Microph	Electronics	79.99
99	Professional Portable Monitor	Electronics	299.99

FIGURE 2: UI

8 Conclusion

We have demonstrated how the Extended Boolean model can enhance the E-commerce product search experience. By integrating textual, field-based, and numeric conditions, as well as Boolean operators, the system can serve user needs more accurately. This leads to more efficient product discovery, improved user satisfaction, and potentially higher sales conversion.

9 Future Work

Future enhancements may include:

- Ranking results based on partial matches.
- Adding fuzzy matching and synonyms.
- Incorporating advanced filtering like date ranges or user reviews.