Софийски университет „Св. Кл. Охридски”

Факултет по математика и информатика

Курсов Проект

на тема: „Система за семантично търсене и извличане на продукти в онлайн магазин”

Студент: **Явор Йорданов Чамов**

Ф.Н. 4MI3400787

Курс: „1“, Учебна година: 2025/26

Преподаватели: **проф. Иван Койчев**

Декларация за липса плагиатство:

- Плагиатство е да използваш, идеи, мнение или работа на друг, като претендираш, че са твои. Това е форма на преписване.
- Тази курсова работа е моя, като всички изречения, илюстрации и програми от други хора са изрично цитирани.
- Тази курсова работа или нейна версия не са представени в друг университет или друга учебна институция.
- Разбирам, че ако се установи плагиатство в работата ми ще получа оценка “Слаб”.

13.2.26 г. Подпис на студента: 

Съдържание

1 Увод

Целта на настоящия проект е разработката на система за търсене на продукти в контекста на електронната търговия. За разлика от стандартните подходи, разчитащи единствено на съвпадение на ключови думи, предложената система интегрира методи за обработка на естествен език, семантично обогатяване и хибридно индексиране (*TF-IDF* и *Elasticsearch*). Системата адресира проблема с "лексикалната пропаст", където заявката на потребителя и описанието на продукта използват различни термини за едно и също понятие. Чрез използване на векторни представяния и синтактичен анализ, решението позволява на потребителите да задават сложни заявки на естествен език (напр. "евтини маратонки за бягане под 100 евро.").

2 Преглед на методите за извличане на информация и семантично търсене

2.1 Подходи и методи за решаване на проблема

Търсенето на продукти в електронната търговия е класическа задача, която еволюира от просто съвпадение на ключови думи към разбиране на потребителското намерение.

Лексикалният подход, базиран на ключови думи е най-традиционният метод, при който документите и заявките се представят като "торба от думи".

- *TF-IDF* (Честота на термините - обратна честота на документите): Статистически метод за оценка на важността на дума в документ спрямо целия корпус. Той намалява тежестта на често срещани думи (като предлози) и увеличава тази на специфичните термини. В настоящия проект този подход е реализиран, използвайки библиотека *scikit-learn*.
- Предимства: Висока бързина, лесна интерпретируемост и ефективност при точно съвпадение на термините.
- Недостатъци: Страда от проблема с речниковото несъответствие – ако потребителят търси "колело", а продуктът е описан като "велосипед", връзката се губи.

Семантичните подходи, базирани на вектори, адресират ограниченията на лексикалното търсене чрез прехвърляне на думите във векторно пространство, където семантично близките думи са разположени близо една до друга.

- *Word2Vec*: Модел, базиран на невронни мрежи, който научава векторни представяния на думите от големи текстови корпуси. В проекта е използван моделът *all-MiniLM-L6-v2* чрез библиотеката *Gensim*.
- Предимства: Улавя контекста и семантиката; повишава пълнотата на търсенето.

За да се разбере структурата на заявката, се използват методи за обработка на естествен език:

- Разбор на зависимости (Синтактичен анализ): Позволява извличане на граматически зависимости между думите. В проекта е интегриран модул, базиран на *spaCy*, който помага за правилното интерпретиране на атрибутите.
- Разпознаване на именовани единици: Идентифициране на същности като Марка, Цена, Тегло и Категория.

2.2 Съществуващи технологични решения

В индустрията съществуват утвърдени платформи за търсене, които предоставят базова функционалност, върху която се надграждат специализирани системи.

Elasticsearch / *Apache Lucene*: Това е най-разпространената търсачка с отворен код. Тя предоставя мощни възможности за пълнотекстово търсене, филтриране и агрегация.

Библиотеки за ОЕЕ (*NLTK, spaCy*): *NLTK* е стандарт за академични цели и предобработка (токенизация, стеминг), докато *spaCy* е оптимизирана за индустриални приложения и сложен синтактичен анализ.

2.3 Сравнителен анализ и обосновка на избрания подход

За целите на курсовия проект е избран хибриден подход, който комбинира силните страни на разгледаните методи.

Характеристика	**TF-IDF (Лексикален)**	**Векторен (Семантичен)**	**Хибриден модел (Настоящ проект)**
Точност	Висока при точни фрази	По-ниска (може да върне само подобни)	**Оптимална** (комбинира точни филтри със семантика)
Пълнота	Ниска (пропуска синоними)	Висока	**Висока** (*Word2Vec* разширява заявката)
Обработка на заявки	Търси само по текст	Търси по смисъл	**Интелигентна** (извлича цена, марка и контекст)
Скорост	Много бърза	Бавна при големи вектори	**Бърза** (благодарение на индексиране)

3 Проектиране на компонентите

Системата е изградена на модулен принцип, като всеки компонент отговаря за специфична част от процеса на извличане на информация. Основните модули са:

- Модул за събиране на данни: За осигуряване на актуален и реалистичен набор от данни е реализиран специализиран модул за автоматизирано извличане на информация от уеб страници.
- Модул за предварителна обработка: Почистване и нормализиране на текстовите данни.
- Модул за семантично обогатяване: Добавяне на метаданни и скрити тагове.
- Индексиращ модул: Поддържа два вида индекси – класическо векторно пространство и *Elasticsearch*.

- Езиков модул за разбиране на заявката: Разбиране на потребителските намерения, извличане на именувани същности и синтактичен анализ.
- Търсещ механизъм: Оркестратор, който комбинира резултатите и ги ранжира.

3.1 Архитектура на системата (Многослоен модел)

Системата е проектирана върху трислойна архитектура.

Слой данни: Отговаря за съхранението на суровите данни и индексите. Тук оперират модулът за събиране на данни и индексиращият модул.

Слой бизнес логика: Съдържа основната функционалност на търсачката. Включва OEE модула, Семантичното обогатяване и Търсещия механизъм. Тук се случват процесите по векторизация и изчисляване на косинусова близост.

Представителен слой: Осигурява взаимодействието с крайния потребител чрез уеб-базиран интерфейс. Интерфейсът е изграден на *Python* с използване на библиотеката *Flask*. Тя обслужва *HTTP* заявките и маршрутизира потребителските действия.

3.2 Модел на данните и представяне на знанията

Всеки продукт в системата се представя като структуриран обект (документ), който обединява лексикална и семантична информация. Схемата на данните включва:

Поле	**Тип**	**Описание**
id	Integer	Уникален идентификатор на продукта.
title	String	Оригинално заглавие от източника.
description	Text	Пълно текстово описание на продукта.
price	Float	Числова стойност на цената за филтриране и сортиране.
brand	String	Марка, нормализирана и извлечена чрез OEE алгоритъм.

Поле	Тип	Описание
semantic_tags	List[String]	Автоматично генерирани етикети (напр. "budget", "lightweight") на база правила.
dependency_features	String (Text)	Синтактични връзки и съставни термини (напр. "mountain_bike", "seat_for_child"), извлечени чрез разграничаване на контекста.

Таблица 1. Модел на данните и описание на атрибутите на продукта.

4 Реализация, тестване/експерименти

4.1 Модул за събиране на данни от уеб страници

За осигуряване на актуален и реалистичен набор от данни е реализиран специализиран модул за автоматизирано извличане на информация от уеб страници. Този компонент позволява изграждането на продуктов корпус по зададени от потребителя параметри на търсене

Модулът използва следните библиотеки за обработка на заявки и HTML съдържание:

- **Requests:** За изпращане на HTTP GET заявки към целевия онлайн магазин.
- **BeautifulSoup (bs4):** За синтактичен анализ на HTML структурата и навигация в дървото.
- **Pandas:** За структуриране на извлечените данни в табличен вид и експортирането им във формат CSV.

Дефинира специфични HTTP заглавия (User-Agent, Accept-Language), които симулират заявка от стандартен уеб браузър. Това е необходимо за избягване на автоматични блокировки от страна на сървъра. Интегрирано е изкуствено забавяне с произволен интервал между заявките. Това предотвратява претоварването на сървъра и намалява риска от засичане на бота.

4.2 Предварителна обработка на данни

Този етап е критичен за намаляване на размерността на речника и шума в данните.

- Токенизация и почистване: Използва се методът clean_text, който премахва специални символи и превръща текста в малки букви.
- Премахване на стоп-думи: Елиминират се често срещани думи без семантична тежест (съюзи, предлози) чрез библиотека *nltk*.

- Свеждане до корен: Прилага се SnowballStemmer, за да се сведат думите до техния корен, което подобрява обхвата на търсенето.

Важен аспект от архитектурата е, че системата обработва не само статичните данни (продуктите), но и динамичните потребителски заявки чрез метода preprocess_enhanced_query. Този метод решава специфичен проблем при хибридното търсене: необходимостта от едновременно прилагане на лингвистична нормализация (Stemming) и запазване на точните семантични етикети.

Токени, съдържащи символа "долна черта" (напр. mountain_bike, bike_with_child_seat), се идентифицират като специални етикети, генерирани от модулите за синтактичен анализ или семантично обогатяване. Тези токени се запазват в оригиналния си вид, без да бъдат подлагани на стеминг, за да се гарантира точно съвпадение с индекса.

Всички останали думи от заявката преминават през стандартния поток за почистване и стеминг (напр. "looking" се превръща в "look").

4.3 Семантично обогатяване

Класът SemanticEnricher (semantic_enrichment.py) реализира логика за добавяне на контекст, който липсва в оригиналното описание.

- Логика: Методът enrich_product анализира структурирани атрибути (цена, спецификации) и добавя текстови етикети.
- Пример: Ако цената е под определен праг, се добавя етикет (бюджетен). Това позволява на потребителя да търси по абстрактни понятия като "евтин", въпреки че думата не фигурира в описанието на продукта.

Неговата основна задача е да реши проблема с "лексикалното разминаване" (vocabulary mismatch), като автоматично генерира скрити метаданни (тагове) на базата на структурираните атрибути на продукта. Този процес превръща количествените характеристики (цена, тегло) и кратките категории в богат набор от синоними и описателни прилагателни.

Методът анализира полето price_numeric. Вместо просто да индексира числото, алгоритъмът го класифицира в ценови диапазон (напр. budget, mid_range, premium) и асоциира съответния набор от синоними от конфигурацията PRICE_TAGS.

Аналогична логика се прилага за атрибута weight. Числовата стойност се преобразува в качествени характеристики чрез речника WEIGHT_TAGS.

Третият етап използва речника category_mappings, за да добави контекст към кратките имена на категориите. Това помага за улавяне на синоними и свързани дейности.

4.4 Индексиране и Векторни модели

1. TF-IDF Индексиране: Реализирано в indexer.py чрез класа TFIDFIndexer. Използва се TfidfVectorizer от библиотеката scikit-learn. Метод build_index създава матрица документ-термин, където тежестта на всеки термин се изчислява по формулата TF-IDF. Това позволява да се даде приоритет на специфичните за продуктите думи пред общите термини. Изчисляването на близост се извършва чрез косинусова близост в метода get_vector.
2. Elasticsearch интеграция: За осигуряване на мащабируемост и бързодействие е реализиран ElasticsearchIndexer (elasticsearch_indexer.py). Методът create_index дефинира полета като name, description, price и векторни полета за embeddings. Методът search използва търсене в цял текст и заявки, позволявайки търсене в множество полета едновременно с различна тежест.
3. Векторни представяния: Класът SemanticEmbeddings (word_embeddings.py) използва библиотеката gensim за обучение на Word2Vec модел. Това позволява улавяне на семантична близост между думите (напр. "маратонка" и "обувка").

Сърцевината на семантичното търсене е реализирана чрез метода за изчисляване на близост (compute_similarity), който трансформира текстовата информация в математически величини. Процесът започва с векторизация (Encoding), при която както потребителската заявка, така и корпусът от документи се подават на предварително обучения модел (Sentence Transformer).

Чрез извикването на self.model.encode, текстовете се преобразуват в плътни вектори (dense vectors) с фиксирана размерност (напр. 384 измерения за all-MiniLM-L6-v2). Критичен момент в реализацията е параметърът normalize_embeddings=True. Той извършва L2 нормализация на векторите, привеждайки ги до еденична дължина. Това е важно оптимизационно решение, тъй като при нормализирани вектори косинусовата прилика е еквивалентна на тяхното скаларно произведение, което е изчислително по-ефективно.

След получаването на векторните представяния, системата изчислява матрицата на приликите чрез функцията cosine_similarity. Тя съпоставя вектора на заявката (query_embedding) с матрицата от вектори на документите (doc_embeddings). Резултатът е масив от стойности в интервала [-1, 1], където стойност, близка до 1, индикира висока семантична близост (сходен смисъл), а стойности около 0 или по-ниски показват липса на връзка.

4.5 Обработка на естествен език и анализ на заявки

Класът NLPParser обработва входната заявка на потребителя. Методът extract_price_constraint използва регулярни изрази, за да разпознае ограничения като "под 100 евро." или "над 50". Методът extract_brand сравнява токените срещу индексиран речник от известни марки.

Методът extract_brands извършва идентификация на търговски марки в текста. Използва се хибриден подход, който комбинира предварително дефиниран списък от известни брандове (KNOWN_BRANDS) с динамично добавени такива. Търсенето се реализира чрез регулярни изрази с граници на думите, за да се гарантира точно съвпадение и да се избегнат грешки при частични съвпадения.

Критичен елемент за точността на системата е методът resolve_ambiguous_terms. В областите често се срещат думи с двойно значение. Например, думата "Giant" може да бъде марка велосипеди или прилагателно ("голям"). Алгоритъмът анализира контекста – съседните думи в заявката – за да определи правилното значение. Ако след "Giant" следва дума като "bike" или "mtb", терминът се класифицира като Бранд; в противен случай се третира като обикновено прилагателно.

Извличане на числови ограничения (Constraint Parsing) За да поддържа заявки от типа "колело под 500 евро" или "по-леко от 12 кг", класът имплементира методите extract_price_constraints и extract_numeric_constraints. Те използват набор от гъвкави Regex шаблони (PRICE_PATTERNS), които разпознават различни фрази от естествения език ("under", "over", "between", "lighter than"). Извлечените данни се нормализират до числови интервали за филтрация (min/max).

Главният метод parse_query обединява всички гореописани стъпки. След като извлече марките, цените и теглото, той ги премахва от оригиналния текст, за да генерира т.нар. clean_query. Тази изчистена версия съдържа само семантичното ядро на продукта (напр. "mountain bike") без шумовите думи за ограниченията.

За по-сложни заявки се използва DependencyParser (dependency_parser.py), базиран на библиотеката spaCy. Анализира граматическата структура на изречението. Това помага да се разбере коя характеристика за кой обект се отнася (напр. в заявка "велосипед с рамка от алуминий", системата разбира, че "алуминий" е атрибут на "рамка", а не на "велосипед" директно). Извличат се именни фрази за по-прецизно търсене.

Методът analyze_query_structure в класа DependencyParser представлява ядрото на синтактичния модул. Неговата основна цел е да премине отвъд

повърхностното разделяне на думи (tokenization) и да изгради пълна граматическа картина на потребителската заявка. Използвайки езиковия модел на spaCy, методът първоначално парсва входния низ в обект тип Doc, който съдържа пълното дърво на зависимостите (dependency tree).

След първоначалния парсинг, алгоритъмът агрегира резултатите от специализирани под-методи в единна структура от данни. Извличат се именни фрази (noun_phrases) и съставни термини (compound_terms), което позволява на търсачката да третира словосъчетания като "mountain bike" като единна концепция, а не като две отделни думи. Също така се анализират предложните връзки (relationships), които дефинират контекста на търсенето (напр. връзката bike -> for -> child указва предназначението на продукта).

Крайният изход е структуриран речник (Dictionary), който служи като "семантична карта" на заявката. Той обединява всички извлечени лингвистични характеристики, включително именувани същности (entities) и ключови концепции.

4.6 Търсеща логика

Класът ProductSearchEngine (search_engine.py) обединява всички компоненти.

Обработка на заявка: Входният текст се подава на OEE парсера за извличане на структурирани данни (цена, марка, и др.).

Филтриране: Ако са открити твърди ограничения (напр. цена < 50), списъкът с кандидати се филтрира предварително.

Ранжиране (Ranking): Системата поддържа две стратегии за оценка на релевантността, в зависимост от избраната конфигурация:

Локален режим: Изчислява се косинусова прилика между векторите на заявката и документите, базирани на TF-IDF тегловна схема.

Elasticsearch режим: Прилага се хибридно ранжиране, което комбинира:

BM25: Вероятностен алгоритъм за текстово търсене, който подобрява стандартния TF-IDF чрез насищане на честотата на термините и нормализация на дължината на документа.

Семантично векторно търсене: Използва се трансформър моделът all-MiniLM-L6-v2 (от библиотеката SentenceTransformers), който преобразува текста в 384-измерни вектори. Това позволява на системата да намира съвпадения по смисъл, дори при липса на общи ключови думи.

4.7 Реализация/Провеждане на експерименти

Цел на експеримента е да се демонстрира способността на системата да обработва кратки потребителски заявки чрез разширяване на контекста и комбинирано ранжиране.

Конфигурация:

- use_elasticsearch=True: Използване на Elasticsearch за базово търсене и съхранение.
- use_dependency_parsing=True: Включен синтактичен анализ за съставни термини.
- use_embeddings=True: Включени векторни представяния за намиране на синоними.

Входна заявка: "Trek mountain bike"

Системата преминава през няколко етапа на трансформация на входния текст, за да максимизира шанса за намиране на релевантен резултат:

1. **Извличане на същности:**

- NLP модулът разпознава "Trek" като известна марка.
- **Действие:** "Trek" се премахва от текстовата заявка и се превръща в **твърд филтър** (Brand: Trek). Това гарантира, че резултатите ще бъдат само от този производител.

2. **Синтактично разширяване:**
   - Остатъкът от заявката е "mountain bike".
   - Парсерът идентифицира, че тези две думи образуват съставен термин.
   - **Резултат:** Добавя се токенът mountain_bike. Това позволява на системата да търси точната фраза като единна концепция, а не просто две отделни думи.
3. **Семантично разширяване:**
   - Чрез векторния модел (Word2Vec/SentenceTransformers) се намират думи, близки по смисъл до "mountain bike".
   - **Добавени термини:** climbing, steep, uphill, mtb.

Забележка: Тези думи не присъстват в оригиналната заявка, но са логически свързани с карането на планински велосипед.

Крайна обработена заявка: mountain bike mountain_bike climb steep uphil mtb

2. Анализ на резултатите (Ranking & Scoring)

Системата връща продукт: **"Electric Mountain Bike E-Bike"**.

Защо този продукт е намерен?

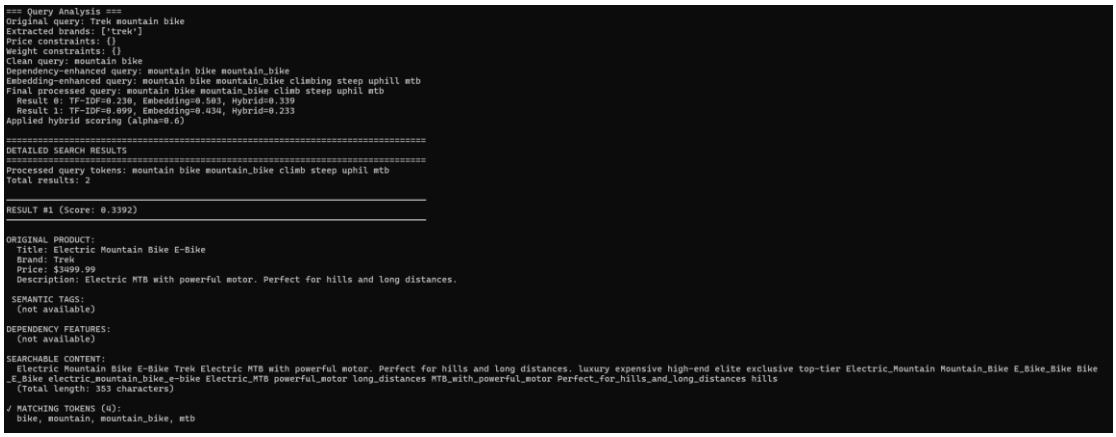
В полето SEARCHABLE CONTENT на продукта виждаме съвпадения, които не биха били възможни при просто търсене:

- **Директни съвпадения:** Думите "Mountain" и "Bike" присъстват в заглавието.
- **Синтактично съвпадение:** Токенът Mountain_Bike (генериран при индексирането на продукта) съвпада с mountain_bike от заявката.

Семантично съвпадение: Описанието на продукта съдържа "Perfect for hills". Въпреки че потребителят не е написал "hills", разширената заявка съдържа climb, steep, uphill, които са семантично близки до "hills" във векторното пространство. Описанието също съдържа MTB, което съвпада с разширената заявка.

За да се гарантира висока точност (Precision) при специфични за домейна термини, системата използва **експертно дефиниран речник от синоними**, реализиран във файла config.py чрез функцията build_semantic_mappings.

В текущата версия на системата, тези речници са **статични**. Това архитектурно решение е избрано с цел детерминистичен контрол върху ключови продуктови характеристики, които не бива да зависят от вероятностни модели.



Фигура 1. Конзолен изход, демонстриращ процеса на семантично разширяване на заявката

Крайният резултат (Score: 0.3392) се изчислява като претеглена сума от текстовото съвпадение (TF-IDF/BM25) и векторната близост (Embedding Similarity).

$$Score = (\alpha \times \text{TF-IDF}) + ((1 - \alpha) \times \text{Embedding})$$

$$Score = (0.6 \times 0.230) + (0.4 \times 0.503)$$

$$Score = 0.138 + 0.2012 = \mathbf{0.3392}$$

5 Заключение

Разработената система успешно демонстрира приложението на съвременни IR и NLP техники в домейна на извличането на информация. Чрез комбиниране на детерминистични правила (за цени, марки, категории и др.) с вероятностни модели (TF-IDF, векторно пространство), се постига баланс между точност и семантично разбиране. Използването на Elasticsearch осигурява необходимата производителност за реално приложение, а модулната архитектура позволява лесно разширяване на функционалността.

В текущата реализация системата разчита на статични семантични мапинги (дефинирани в config.py) и предварително обучени векторни модели. Въпреки че този подход е ефективен за начален етап той не отчита еволюцията на езика и

специфичния жаргон на потребителите във времето. Затова, ключово направление за развитие е внедряването на динамични адаптивни речници.

6 Използвана литература и източници

Luminati-io. eCommerce-dataset-samples. GitHub Repository. Достъпен на: https://github.com/luminati-io/eCommerce-dataset-samples. [Последен достъп: Февруари 2026]