Case Study - Data Science

Anna Marie Minarovičová

September 19, 2024

Architektura vyhledávacího enginu

Pojďme se nejprve zamyslet, jakým problémům budeme čelit při zpracování daného problému. Zkusme si odpovědět na tři základní otázky nutné k vybudování architektury vyhledávacího enginu.

Nastiňme také základní implementaci v Pythonu, jež je vhodný z toho důvodu, že dokáže řešit všechny backendové části zadaného problému (zpracování dat, zpracování user query, vytvoření ML a AI modelů). Python není nejrychlejší programovací jazyk, neparalelizuje a pracuje s omezenou pamětí počítače. Pokud by tedy nastala situace, kdy by data eshopu měla obrovský objem, narazili bychom na limitace a museli bychom se zamyslet nad použitím jiné techniky než Pythonu. Předpokládám ale, že v eshopu nebudou mít jako produkty statisíce položek, tedy si myslím že použití je oprávněné.

Struktura pro uchovávání produktů v eshopu

Máme data v CSV formátu s 6 proměnnými *id, název, popis, kategorie, cena, recenze*. Načtěme je do DataFramu v Pandas. Pojmenujme DataFrame například *products*. Tato tabulková struktura je velmi praktická pro následné zpracování.

Pro vytvoření modelu si potřebujeme uvědomit, které kategorie nám nesou relevanci vzhledem k user query. v tomto případě je to *název, popis, kategorie*, jelikož hodnocení a cena jsou numerické proměnné, které přímu se stringovým dotazem nesouvisí. Spojme tedy tyto tři proměnné do jedné a přidejme ji jako další proměnnou "popisujici_slova" do DataFramu products (*products["popisujici_slova"] = products["název"] + " " + products["popis"] + " " + products["kategorie"] + " " + products["kategorie"] + " " + products["kategorie"] + " " + products["kategorie"]*). Důvod vícenásobného přidání kategorie je ten, že bychom ji chtěli prioritizovat. Pokud se nám ve query objeví název kategorie, uživatel pravděpodobně nebude zvědavý na výsledky z jiné kategorie, tedy takto jí přidáme váhu.

Nyní proveďme zpracování jazyka u této proměnné. Udělejme následující:

- tokenizace: rozdělení stringu na list obsahující jednotlivá slova, *query.split(" ")*
- stemming: převedení slov do základních tvarů, klasicky v angličtině se používá Porterův algoritmus, v češtině by podle internetu měla fungovat knihovna NLTK v Pythonu, nebo software Snowflake
- odfiltrování stop words a interpunkce pro redukci celkové velikosti slovníku: části, jež nenesou význam (v tomto případě obecně zájmena, slovesa a nesklonné slovní druhy).
- list slov nakonec spojme do jednoho listu *" ".join(popisujici_slova)*.

K tomuto procesu můžeme použít například knihovnu *stanza*, která podporuje i český jazyk. Stop words musíme dát ve formě listu. Pokud bychom ale například pracovali s angličtinou, můžeme použít knihovnu nltk, která obsahuje i balíček stop words.

Aktuálně máme DataFrame se sedmi proměnnými.

Porozumění user query a vrácení relevantních výsledků

Prvním úkolem je, pokud máme user query, navrátit uživateli relevantní produkty. Mějme string user query, kterou získáme z vyhledávacího pole na stránkách po aktivaci vyhledávacího tlačítka. Pro zjednodušení předpokládejme, že uživatel neudělal ve svém dotazu pravopisnou chybu (v reálném světě ale uživatelé chyby dělají, proto ideální by bylo implementovat i spelling correction pomocí správného nástroje). Pro zpracování query udělejme to samé, jako u produktů

- tokenizace,
- stemming,
- odfiltrování stop words a interpunkce,
- spojení slov do jednoho stringu.

Nyní máme user query ve tvaru zpracovaného stringu, obsahujícího slova v základním tvaru nesoucí význam, DataFrame obsahující v prvních a poslední proměnné ID produktů a slova je popisující.

Použijme **Vector Space Model** používající tf-idf score pro navrácení relevantních produktů uživateli. Tento model řadí produkty podle jejich možné relevance, detailnější popis například na Wikipedii `https://en.wikipedia.org/wiki/Vector_space_model`. v našem případě termy budou osy mnohodimenziálního prostoru a produkty a query jsou jeho vektory. Blízkost jednotlivých bodů prostoru je měřena pomocí úhlu - cosine similarity, která je vhodnější, než euklidovská vzdálenost.

Nejprve si napišme kód pro inicializaci modelu. Toto bude vypočteno jednou a uloženo v paměti. Budeme používat knihovnu **scikit-learn**, jež obsahuje efektivní implementaci Vector Space Modelu a výpočtu cosine similarity.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Vytvoreni korpusu popisu produktu
korpus = products["popisujici_slova"].tolist()

# Vypocet TF-IDF matice pro produkty
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(korpus)
```

Zpracování query.

```
# Vytvoreni vektoru dotazu pomoci stejneho TF-IDF vectorizeru
query_vector = vectorizer.transform([dotaz])

# Vypocet kosinove podobnosti mezi vektorem dotazu a~vektory produktu
cosine_similarities = cosine_similarity(query_vector, tfidf_matrix).
    flatten()

```

```
# Ziskani relevantnich produktovych ID a~jejich skore (vysledky, jez nemaji nulove score)
relevantni_produkty = {produkt_id: score for produkt_id, score in zip(products["id"], cosine_similarities) if score > 0}
```

Nyní máme relevantnost query vzhledem k názvu, popisu a kategorii, která nám jednak vyfiltruje relevantní výsledky, za druhé nám je seřadí.

Zatím jsme ovšem do vyhodnocení nepromítli hodnocení produktů uživateli. Pojďme se nyní nad touto vlastností zamyslet. Vyšší hodnocení by drobně mělo přidávat produktu přidávat na relevantnosti. Jednak na uživatele bude působit lépe, když mu budou vraceny dobře hodnocené produkty, za druhé i pro eshop je lepší prodávat věci, se kterými jsou uživatelé spokojeni, protože pak je menší riziko, že si uživatelé k eshopu vybudují "antipatii" a produkty se nebudou tolik vracet ve čtrnáctidenní lhůtě. Nabízelo by se například vynásobit hodnoty navrácené modelem nějakou konstantou či funkcí závislou na hodnocení (pro vyšší hodnocení vyšší multiplikátor a naopak). Nejlepší by bylo asi vyzkoušet několik druhů multiplikátorů a posuzovat, jak moc mají vliv na relevanci. Nesmíme dopustit, aby toto přebilo relevanci. v této oblasti by bylo vhodné udělat ještě větší výzkum.

Použití user analytics pro hodnocení relevantnosti nabízených produktů

Nyní když máme základní model, který nám seřadí výsledky pro zadanou user query a navrátí relevantní výsledky takto seřazené. Musíme ovšem vzít v úvahu dvě skutečnosti, a to:

- model není dokonalý
- uživatelé nejsou dokonalí.

Co bych tím chtěla říci?

Model je pouze nástroj, který uživatelům inherentně nerozumí jako člověk. v jazyce existuje mnoho nejasností, kvůli kterým může mít model problém s porozuměním uživatelově dotazu. Zároveň lidé občas mají také problém přesně definovat dotazy, kdy uživatelova information need a query se mohou výrazně lišit.

Chtěli bychom tedy použít relevance feedback od skutečných uživatelů k tomu, abychom vylepšili náš dosavadní vyhledávací model.

V kontextu e-commerce, kdy uživatel zadá dotaz a zobrazí se mu výsledky, můžeme sledovat, na které produkty klikne a které ignoruje. Produkty, které si uživatel prohlíží, označujme jako relevantní, a ty, které ignoruje, považujeme za méně relevantní pro daného uživatele. S pomocí Rocchiova algoritmu můžeme tyto informace použít k optimalizaci modelu, aby v budoucnu poskytoval přesnější výsledky.

Rocchiův algoritmus upravuje původní dotaz na základě produktů, které uživatel označil jako relevantní nebo nerelevantní. Klíčovou myšlenkou je, že relevantní produkty přibližují dotaz k tomu, co uživatel skutečně hledá, a nerelevantní produkty jsou použity k odklonění dotazu od těchto nerelevantních informací.

Algoritmus se definuje vzorcem

$$\mathbf{q}_{\text{new}} = \alpha \mathbf{q}_{\text{original}} + \beta \frac{1}{|D_r|} \sum_{\mathbf{d_j} \in D_r} \mathbf{d_j} - \gamma \frac{1}{|D_{nr}|} \sum_{\mathbf{d_k} \in D_{nr}} \mathbf{d_k}$$

kde:

- $\mathbf{q}_{\text{new}}$ je nový vektor dotazu.

- $\mathbf{q}_{\text{original}}$ je původní dotaz.
- D_r je množina relevantních produktů.
- D_{nr} je množina nerelevantních produktů.
- α, β, γ jsou váhy, které upravují důležitost původního dotazu, relevantních produktů a nerelevantních produktů.

Když uživatel zadá novou query, Rocchiův algoritmus se uplatní tím, že dotaz posuneme směrem k relevantním produktům z minulých interakcí a odkloníme jej od nerelevantních. Tento posun v dotazu zlepší přesnost budoucích výsledků tím, že model přiblíží výsledky tomu, co uživatel skutečně hledá.

Jak to bude přesně fungovat? Mějme novou user query od uživatele, od nějž již máme nějaký relevance feedback. Poté:

- převedeme novou query do vektorového prostoru,
- na základě uživatelské zpětné vazby z minulých interakcí určíme produkty, které jsou relevantní, a ty, které jsou nerelevantní,
- použijeme Rocchiův algoritmus k úpravě původního dotazu. Přidáme vážený průměr vektorů relevantních produktů a odečteme vážený průměr nerelevantních produktů,
- nová user query bude použita k vyhledání produktů, což zvyšuje pravděpodobnost, že uživatel dostane relevantnější výsledky.

V Pythonu implementujme Rocchiův algoritmus pomocí knihovny NumPy, která poskytuje nástroje pro efektivní práci s maticemi a vektory.

Doporučování produktů - "Doporučeno pro Vás" a "Uživatelé také koupili"

Předpokládejme, že dovedeme vrátit relevantní výsledky pro zadanou user query pro daného uživatele. Nyní se pokusme vyřešit problémy doporučení. Pokud známe chování uživatele (které produkty si otevřel, koupil), dokážeme mu efektivně doporučit další produkty, které nakoupit? a pokud se dostane na stránku s určitým produktem, dovedeme mu doporučit další produkty, které zakoupili lidé, co zakoupili i tento produkt?

Při řešení problému, jaké produkty nabízet uživateli, na základě jeho aktivity, budeme postupovat následovně:

- Vezmeme vektory produktů, které uživatel v poslední době prohlížel nebo koupil. Řekněme, že posledních deset produktů. Tyto produkty reprezentujme jako body v mnohodimenzionálním prostoru jako dříve.
- Rozdělme si produkty podle kategorií.
- Pro každou kategorii vypočítáme centroid (`https://personal.utdallas.edu/~herve/abdi-WireCS-Centroid-2009.pdf`), který nám nějak reprezentuje, kde leží uživatelův zájem v dané kategorii.
- Pomocí *cosine similarity* vypočítáme podobnost mezi těmito centroidy a ostatními produkty ve vektorovém prostoru.
- Vyberme celkově nejbližší produkty k našim centroidům, které nebyli mezi původními prohlíženými produkty a tyto zobrazme.

Tento přístup zajistí, že uživateli budou doporučeny produkty, které jsou z hlediska obsahu (popisu, vlastností) podobné těm, které uživatel považoval za zajímavé. Například, pokud si uživatel prohlížel notebooky s určitými parametry (procesor, paměť, cena), systém mu doporučí jiné notebooky s podobnými parametry.

Počet deset pro poslední zobrazené produkty a deset pro doporučené je odhad. Samozřejmě by to bylo nutné vyzkoušet s více různými konstantami a dívat se, jak se doporučování bude chovat.

V Pythonu používáme při výpočtu model a principy, které jsme již představili (scikit-learn, NumPy, Pandas).

Pokud chceme doporučit produkty, které si zakoupili jiní uživatelé, kteří také zakoupili produkt, který uživatel právě prohlíží, musíme si u každého produktu uchovávat ID produktů, které byly společně s ním zakoupeny. Poté by nám již pouze stačilo se podívat na například pět nejčastějších ID a jim korespondující produkty zde uvést. Předpokládám, že by zde šla udělat ještě nějaká personalizovaná optimalizace.

Clustering

Pokud bychom u našich dat produktů chtěli zavést automatickou klasifikaci do kategorií, můžeme použít kombinaci Vector Space Model a **Support Vector Machine** modelu. Máme produkty jako vektory a známe jejich klasifikaci. Na těchto datech můžeme model natrénovat. Je potřeba je rozdělit data na tréninkovou a testovací sadu. Použijme poměr 80:20. V Pythonu můžeme použít funkci *train_test_split* z knihovny *sklearn.model_selection*. Po rozdělení dat, můžeme začít trénovat Support Vector Machine model. SVM se snaží najít optimální hyperrovinu, která odděluje různé kategorie produktů (`https://en.wikipedia.org/wiki/Support_vector_machine`). V Pythonu můžeme opět použít knihovnu sklearn. Model natrénujeme na tréninkových datech, a poté vyhodnotíme model na testovacích datech, při zkontrolování accuracy a precision. Pokud by nám některá z metrik přišla příliš nízká, můžeme se pokusit zlepšit model pomocí například hyperparameter tuning pomocí *GridSearchCV*. Když máme model natrénovaný, můžeme jej použít ke klasifikaci nově přidaných produktů, jejichž polohy ve vektorovém prostoru nám udávají pouze veličiny název a popis.

Zlepšení

Jaký bude interface

Frontend tohoto vyhledávacího enginu je velmi jednoduchý. Potřebujeme jedno vyhledávací pole, kam uživatel zadá dotaz ve formě řetězce a stiskne tlačítko “search”. Poté se pod vyhledávacím polem zobrazí seřazené relevantní výsledky. Na spodní části stránky mohou být personalizované doporučené produkty pro uživatele, u kterých máme jejich statistiky, a také nejprodávanější produkty za určité období (například měsíc) pro nové uživatele. Velmi jednoduchým rozhraním pro vytvoření uživatelského rozhraní je například Bootstrap.

Cachování

Dá se očekávat, že dotazy zadávané do eshopu se budou v průběhu času opakovat. Je jen konečné množství způsobů, jak se uživatel může zeptat například na mobilní telefon. Zpracované user query se tedy dají cachovat, což může urychlit vyhledávání.

Přidání relevantnosti hodnocení

Data neobsahují počet zakoupení ani počet hodnocení daného produktu. Tyto dvě velmi důležité metriky by bylo dobré sledovat. Například číselné hodnocení je takto mnohem méně relevantní. Pokud máme produkt hodnocen pěti hvězdičkami jen jedním uživatelem a pěti hvězdičkami stovkami uživatelů, budeme mít větší informaci o spokojenosti uživatelů u druhého produktu. Tedy počet hodnocení by bylo dobré přidat k hodnocení jako váhu. Počet zakoupení (oblíbenost) by zase dávalo smysl, aby produktu přidával na relevantnosti.

Eshop přeprodávající produkty třetí strany

Navržený model svým vážením předpokládá, že se jedná o eshop, kdy jsou specifikace produktů "v pořádku", jelikož je senzitivní na frekvenci výskytu slov. Pokud bychom měli popis produktu "televize, televize, televize, televize", relevantnost tohoto produktu pro jakoukoli query obsahující "televize" by byla uměle zvýšena. Model je tedy vyhovující pro interní eshopy, kdy mohou být popisy korigovány, ne eshopy přeprodávající produkty třetích stran.

Spelling correction

Jak již bylo ¨zmíněno, pro user query by bylo dobré implementovat spelling correction. Na to existují například Python knihovny TextBlob, JamSpell nebo Autocorrect, obsatující jazykové modely na korekci. Nejsem si ovšem jista jejich funkčností v češtině, bylo by tedy nutné se zamyslet, zda-li by použít některou z nich, popřípadě kterou.