Dokumentácia- Zadanie 2

Bc. Adrián Vyskoč

https://github.com/adrianvyskoc/vinf-recommender

Obsah

Úvod

Mojou úlohou bolo vytvoriť odporúčací systém, ktorý bude odporúčať jednotlivým zákazníkom produkty, ktoré by ich mohli zaujímať na základe ich predošlého správania na webe. Dáta ktoré sme dostali hovoria o interakciách zákazníkov s produktami.

Na riešenie tohto problému som si vybral metódu kolaboratívneho filtrovania – memory based – item based.

Analýza dát

K dispozícii máme dva datasety:

- **events_train.csv – 14 614 385 záznamov**
  Tento dataset obsahuje zoznam eventov, ktoré sú typu view_item, teda zákazník si pozrel daný produkt.
- **purchases_train.csv – 188 712 záznamov**
- Tento dataset obsahuje zoznam eventov, ktoré sú typu purchase_item, teda zákazník si kúpil daný produkt.

Oba tieto datasety obsahujú nasledovné atribúty:

- customer_id (string)
- timestamp (string)
- event_type (string)
- product_id (string)
- title (string)
- category_name (string)
- price (float)

customer_id
Identifikátor zákazníka, môže obsahovať duplikáty.

timestamp
Časová pečiatka, kedy daný zákazník interagoval s produktom.

event_type
Typ, akým spôsobom daný zákazník interagoval s produktom. Môže byť dvoch typov:

- nákup (purchase_item)
- prezretie produktu (view_item)

product_id
Identifikátor produktu. Môže obsahovať duplikáty.

title
Názov produktu, s ktorým zákazník interagoval. V datasete events_train.csv chýba 14 612600. V datasete purchases_train.csv chýba 188684 eventom tento atribút.

category_name
Kategória daného produktu. V datasete purchases_train chýba 143 814 eventom tento atribút. V datasete events_train.csv chýba 11 223 551 eventom tento atribút.

price
Cena daného produktu.V datasete purchases_train.csv chýba 44886 eventom tento atribút.

Predspracovanie dát

Ako prvú vec, ktorú som vykonal, keď som si načítal datasety (events, purchases) bolo že som tieto datasety spojil do jedného. Následne som tieto datasety zoradil podľa časovej pečiatky od najmenšieho po najväčší.

Čistenie dát

Jediné čistenie dát, ktoré som vykonal bolo, že som odstránil stĺpce, ktoré som ďalej nepotreboval – price, category.

Následne som v stĺpci event_type nahradil hodnoty nasledovne:

- Hodnotu *view_item* som nahradil hodnotou *1*
- Hodnotu *purchase_item* som nahradil hodnotou *2*

```
df['event_type'] = df['event_type'].replace({
      'view_item': 1,
      'purchase_item': 2
})
```

Toto nahradenie je spravené nasledovne preto, aby som odzrkadlili relevantnosť jednotlivých typov eventov, teda že *purchase_item* má vyššiu váhu ako *view_item*.

Zakódovanie identifikátorov

Aby som dáta mohol pohodlne vložiť do sparse matice, je nutné identifikátory *customer_id* a *product_id* zakódovať do numerických hodnôt. Toto som vykonal pomocou **LabelEncoder** z knižnice **sklearn.preprocessing**.

```
le_customer = preprocessing.LabelEncoder()
le_product = preprocessing.LabelEncoder()

df['customer_id'] = le_customer.fit_transform(df['customer_id'])
df['product_id'] = le_product.fit_transform(df['product_id'])
```

Rozdelenie na trénovacie a testovacie dáta

Na to aby sme mohli natrénovať a následne overiť vytvorený odporúčací model je nutné rozdeliť celú množinu dát na testovaciu a trénovaciu zložku. Toto som vykonal pomocou funkcie **train_test_split** z knižnice **sklearn.model_selection**.

Rozhodol som sa použiť 80% dát na trénovanie a 20% dát na testovanie.

```
df_train, df_test = train_test_split(df, test_size=0.20, shuffle=False)
```

Tvorba sparse matice

Keďže dáta budú mať veľmi riedky charakter je nutné ich reprezentovať vhodným spôsobom. Načítanie takýchto dát do DataFrame nie je možné kvôli pamäťovému limitu.

Na reprezentáciu takýchto dát som zvolil sparse maticu z knižnice **scipy.sparse**. Najprv som vytvoril maticu COO, ktorú som následne zkonvertoval do matice typu CSR (Compressed Sparse Row). Táto matica ponúka efektívnu prácu s riadkami a keďže máme ako index zvolené *product_id*, tak táto matica je vhodná.

```
mat_coo = coo_matrix(
    (
        np.array(df_train['event_type']),
        (
            np.array(df_train['product_id']),
            np.array(df_train['customer_id'])
        )
    )
)
mat_csr = mat_coo.tocsr(copy=False)
```

Opis riešenia

Na vytvorenie odporúčacieho systému som použil algoritmus KNN. Tento algoritmus som si importoval z knižnice **sklearn.neighbors**.

Inicializácia modelu

Model som inicializoval s nasledovnými parametrami:

- **metric**: *cosine*
  Na vypočítavanie kosínovej vzdialenosti medzi jednotlivými pozorovaniami.
- **algorithm**: *brute*
  Algoritmus použitý na výpočet najbližších susedov. Použili sme *brute*, pretože v dokumentácii sa uvádza, že pri „riedkom“ (sparse) vstupe sa aj tak prepíše tento parameter na *brute.*
- **n_neighbors**: *10*
  Počet susedov, respektíve vygenerovaných odporúčaní, ktorý sa použije ak nezadáme koľko chceme odporučiť do funkcie recommend.
- **n_jobs**: *-1*
  Počet paralelných výpočtov, ktoré môžu bežať pri hľadaní susedov. Hodnota *-1* znamená že použitie všetkých procesorov.

Po inicializácii som natrénoval model na trénovacích dátach – na vytvorenej „riedkej“ (sparse) matici.

```
# vytvorenie inštancie KNN
model_knn = NearestNeighbors(metric='cosine', algorithm='brute',
n_neighbors=10, n_jobs=-1)

# natrénovanie na trénovacích dátach
model_knn.fit(mat_csr)
```

Funkcia recommend

Na samotné odporúčanie som vytvoril funkciu **recommend**, do ktorej ako vstup musíme zadať nasledovné parametre:

- model_knn – model, ktorý sme vytvorili
- customer_id – identifikátor používateľa, ktorému chceme odporúčať produkty
- n_recos – počet produktov, koľko chceme odporučiť

Implementáciu funkcie môžeme rozdeliť do nasledovných krokov:

1. Získanie produktov, s ktorými interagoval používateľ v testovacej množine dát.
2. Získanie produktov, s ktorými interagoval používateľ v trénovacej množine dát.
3. V prípade, že daný používateľ nemá žiadne produkty v trénovacej množine, vrátime preňho generické odporúčanie – *n* najpopulárnejších produktov.
4. Samotné generovanie odporúčaní pomocou algoritmu KNN. Odporúčania sa generujú metódou **item-based** tak, že pre každý produkt, s ktorým používateľ interagoval v trénovacej množine vygenerujeme *n* (n_recos) odporúčaných produktov.
5. Vytvorenie mapy (dict) s najmenšími vzdialenosťami pre každý produkt. Tento krok sa vykonáva, lebo pre jednotlivé produkty z trénovacej množiny s ktorými interagoval daný používateľ nám môže KNN vrátiť aj rovnaké produkty ale s inou vzdialenosťou. Treba však zobrať tú vzdialenosť, ktorá je najmenšia.
6. Z mapy najlepších odporúčaní vyberieme *n* najlepších (tie s najmenšou vzdialenosťou).
7. Výpočet presnosti (precision) pre daného používateľa. Ako som počítal presnosť si môžete pozrieť v sekcii Vyhodnotenie.
8. Výpočet pokrytia (recall) pre daného používateľa. Ako som počítal pokrytie si môžete pozrieť v sekcii Vyhodnotenie.
9. Vrátenie výsledkov (distances, indices/product_ids, precision, recall).

Jednotlivé kroky implementácie funkcie recommend sú zaznačené komentármi v kóde.

Optimalizácia

Skúšal som použiť rôzne parametre pri inicializácii modelu. Hlavný parameter, na ktorý som sa zameral je parameter *metric*, v ktorom určujeme, akým spôsobom sa bude počítať vzdialenosť medzi vzorkami.

Vyhodnotenie

Precision

Presnosť v našom kontexte znamená, koľko % produktov si zákazník reálne pozrel z tých ktoré som mu odporučil. Vzorec je nasledovný:

$$\frac{true\ positives\ (odporúčané\ produkty\ a\ zároveň\ pozreté\ v\ test\ množine)}{(true\ positives + false\ positives)}$$

$$true\ positives + false\ positives$$
$$= \min(počet\ predikovaných\ produktov, počet\ pozretých\ produktov\ v\ test\ množine)$$

Berieme minimum z počtu odporučených produktov a počtu prezretých produktov v test množine, pretože ak odporúčam 10 produktov a daný zákazník si pozrel v test množine len napr. 3, tak ak by sme do menovateľa vybrali 10, nikdy by sme nemohli dosiahnuť 100%. Ak ale vygenerujeme 10 odporúčaní a zákazník si z nich pozrie 3 (pri celkovom počte 3 pozreté produkty v test množine), tak sme dosiahli presnosť 100%.

Presnosť som počítal na jednotlivých zákazníkoch samostatne a následne som tieto hodnoty spočítal a vydelil počtom zákazníkov, čím som dostal celkovú presnosť.

Celkovú presnosť som počítal ako:

$$\frac{\sum presnosť\ pre\ jednotlivých\ používateľov}{počet\ používateľov}$$

Recall

Pokrytie v našom kontexte znamená, koľko % produktov odporučených sa naozaj nachádza v testovacej množine daného zákazníka. Rozdiel oproti presnosti je v tom, že tu neberieme ohľad na počet odporučených produktov.

$$\frac{true\ positives\ (odporúčené\ produkty\ a\ zároveň\ pozreté\ v\ test\ množine)}{\text{true positives} + \text{ false negatives } (počet\ pozretých\ produktov\ v\ test\ množine)}$$

Recall som počítal na jednotlivých zákazníkoch samostatne a následne som tieto hodnoty spočítal a vydelil počtom zákazníkov, čím som dostal celkový recall.

Celkový recall som počítal nasledovne:

$$\frac{\sum pokrytie\ pre\ jednotlivých\ používateľov}{počet\ používateľov}$$

Mean average precision

Mean average precision som počítal tak, že som si vypočítal average precision pre jednotlivé skupiny zákazníkov a následne som z týchto hodnôt vypočítal mean.

Výsledky

Pri použití euklidovskej vzdialenosti (metric='minkowski', p=2) sa presnosť znížila v priemere o 0.05, pri testovaní na vzorke 200 zákazníkoch.
Priemerná presnosť pri našich optimálnych parametroch () na vzorke 200 zákazníkoch bola 1.7% - 3%.

Pri zvýšení počtu zákazníkov na 300 som zaznamenal zvýšenie presnosti o 1% na cca 0.042 (výsledky z tohto pozorovania si môžete pozrieť v súbore results_300).

Priemerná presnosť pri vzorkách po 5000 zákazníkoch bola 0.08, čo je 8% (výsledky z tohto pozorovania si môžete pozrieť v súbore *results_50000*). Priemerné pokrytie v tomto pozorovaní sa pohybovali od 0.074 do 0.08.

Priemerná presnosť pri náhodných 10000 zákazníkoch bola 0.08 % (výsledky si môžete pozrieť v súbore *random_10000 _customers*). Pokrytie pri tomto pozorovaní bolo cca 0.076.

Výhody a nevýhody

+ Jednoduchá implementácia – ľahké pochopenie
+ Zohľadňovanie váhy purchase_item/view_item eventov
+ Výpočtovo nenáročný model
+ Možnosť zvoliť si počet odporučených produktov

– V odporúčaní nie je zahrnutý čas zakúpenia/pozretia produktu
– Nie príliš veľká presnosť

Zhrnutie

Podarilo sa mi vytvoriť odporúčací model, ktorý pracuje pomocou algoritmu KNN.

Počas testovania som skúšal použiť rôzne parametre, z ktorých najlepšie vyšla kombinácia parametrov zo sekcie **Opis riešenia – Inicializácia modelu**. Parameter *algorithm* nemalo zmysel meniť, pretože z dokumentácie vyplýva, že pri sparse dátach sa tento algoritmus aj tak nastaví na hodnotu *brute*.

Pri vyhodnocovaní som si všimol, že pri menšom počte (cca 200) zákazníkov mi vyšla omnoho menšia presnosť (cca o 5%) ako pri vyššom počte (cca 1000).

Takisto som testoval, ako sa bude odporúčač správať, ak zvolím inú reprezentáciu view_item a purchase_item eventu. Ak som použil napríklad reprezentáciu view_item – 1 a purchase_item – 3, presnosť sa znížila v priemere 0.005 na rovnakej vzorke oproti reprezentácii view_item – 1 a purchase_item – 2.