Eksploracja masywnych danych
Projekt – Sprawozdanie I
Sformułowanie problemu i przetwarzanie danych

Tomasz Kuliński Wojciech Mioduszewski

21 listopada 2013

1 Sformułowanie problemu

Konkurs nosi nazwę: "Personalized Web Search Challenge". Trwa do 10 stycznia 2014, a jako nagrodę można otrzymać 5000$, 3000$ lub 1000$ za odpowiednio pierwsze, drugie i trzecie miejsce.

Link: http://www.kaggle.com/c/yandex-personalized-web-search-challenge

1.1 Krótki opis problemu

Zadanie polega na tym aby uporządkować na nowo listę linków zwracanych przez przeglądarkę dla danego zapytania pod konkretnego użytkownika. W tym celu do dyspozycji otrzymujemy dane historyczne przeglądarki oraz obecnej sesji.

1.2 Opis danych

Zbiory treningowy i testowy mają właściwie identyczną postać. Składają się z 3 rodzajów linijek:

- **Dane o sesji** zawierają unikalny identyfikator sesji, numer dnia i identyfikator użytkownika
- **Dane o zapytaniu** zawierają identyfikator zapytania, identyfikator sesji, czas od rozpoczęcia sesji, typ zapytania (normalne lub testowe, występujące tylko w zbiorze testowym), numer zapytania w sesji, listę identyfikatorów termów i listę par identyfikatorów urli i identyfikatorów domen.
- **Dane o kliknięciu** zawierają identyfikator sesji, czas od rozpoczęcia sesji, identyfikator urla i numer zapytania w sesji.

Rozmiar danych przedstawia się następująco:

Typ	Zb. treningowy	Zb. testowy
dane o sesji	34'573'630	797'867
dane o zapytaniu	65'172'853	1'543'096 (797'867 testowych)
dane o kliknięciu	64'693'054	632'539

Table 1: Rozmiar danych

1.3 Opis metody oceny rozwiązań

Podstawowym problemem jest zakwalifikowanie każdego linka do jednej z trzech kategorii: **0 - nieważnych, 1 - ważnych i 2 - bardzo ważnych**. Do pierwszej powinny zaliczać się te linki, które nie zostaną wybrane przez użytkownika, albo wskazywane przez nie strony będą przez niego analizowane krócej niż 50 jednostek czasu. W grupie drugiej mają się znaleźć linki do tych stron, które zostaną wybrane przez użytkownika i przeglądane do 399 jednostek czasu. Wszystkie linki do stron przeglądanych przez przynajmniej 400 jednostek mają się znaleźć w trzeciej grupie.

Zadaniem jest takie przeoragnizowanie kolejności url_id w zapytaniach testowych, aby były posortowane zgodnie z malejącą wartością kategorii tak, aby przedstawić użytkownikowi jak najbardziej stosowne linki. Oceną rozwiązania jest miara Normalized Discounted Cumulative Gain, wyrażająca się wzorem:

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

gdzie: $DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i}-1}{\log_2(i+1)}$, a $IDCG_p$ jest idealną wartością miary.

2 Reprezentacja danych w pamięci zewnętrznej

Dane zostały podzielone na sześć tabel w bazie PostgreSQL. Wszystkie kolumny poza jedną są typu integer. Przyjęto, że klucze główne będą oznaczane podkreśleniem, a klucze obce - *kursywą*.
Pierwsza to tabela **Session**, zawierająca podstawowe informacje o sesji.

- session_id
- day
- user_id

Druga tabela to **Query**, zawiera informacje dotyczące pojedynczego zapytania.

- q_id
- query_id
- *session_id* (Session)

- serpid
- time_passed
- is_test (boolean)

q_id jest sztucznie nadanym kluczem głównym tabeli, query_id jest wartością odczytaną z pliku z danymi. time_passed, podobnie jak w jednej z kolejnych tabel, reprezentuje czas od ostatniej akcji użytkownika (a nie od rozpoczęcia sesji, jak było to zapisane w pliku z danymi).
Trzecia tabela to **Query_term**, przechowująca informację o pojedynczym termie użytym w zapytaniu.

- result_id
- term_id
- query_id
- pos

Wątpliwości może budzić jedynie kolumna pos, będąca pozycją termu w zapytaniu.
Czwarta tabela, **Query_url**, zawiera informacje o pojedynczym rezultacie zapytania.

- result_id
- *url_id* (Url)
- query_id
- pos

Piąta tabela, **Click**, reprezentuje pojedyncze kliknięcie linku przez użytkownika.

- click_id
- *url_id* (Url)
- *q_id* (Query)
- time_passed

Ostatnia tabela, **Url**, zawiera informację o domenie linku.

- url_id
- domain_id

Początkowo schemat miał minimalnie inną postać (tabele Query_term i Query_url zamiast kolumny query_id miały kolumnę q_id, a tabela Click zamiast pary (url_id, q_id) - klucz obcy do Query_url). Podczas pracy nad danymi zorientowaliśmy się jednak, że taka sama wartość query_id może występować w różnych sesjach (okazało się, że jest to wartość unikalna dla danego zestawu termów i adresów url). Ta stosunkowo niewielka zmiana pozwoliła na znaczne zredukowanie wielkości tabel Query_term i Query_url - z 200 mln. i 700 mln. na *jedynie* 80 mln. i 200 mln (oczywiście są to wartości dla zbioru treningowego, zbiór testowy ma te wartości mniejsze o ponad rząd wielkości).

Czas potrzebny na umieszczenie danych w bazie podzieliliśmy na trzy części:

- przepisanie pliku z danymi na sześć plików odpowiadających kolejnym tabelom w bazie danych - 344s (5min 44s)
- właściwe przerzucenie danych do bazy - 3873s (1h 4min 33s)
- założenie indeksów i nałożenie ograniczeń (klucze podstawowe i obce) - 5992s (1h 39min 52s)

3 Reprezentacja danych w pamięci operacyjnej

Wyszukiwanie najbliższych sąsiadów miało polegać na wyznaczeniu 100 najczęściej występujących termów i 100 najczęściej występujących urli, a następnie obliczaniu współczynnika Jaccarda dla poszczególnych zapytań w odniesieniu do wspomnianych termów i urli. Okazało się, że już wyznaczenie 100 najczęściej występujących urli znacznie przerosło możliwości bazy danych - poniższe zapytanie (wykonywane jeszcze na starej wersji schematu danych) zwracające nie do końca poprawne wyniki, było wykonywane przez ponad godzinę.

Listing 1: Zapytanie 1

```
SELECT url_id, COUNT(*) AS c
FROM (
    SELECT DISTINCT ON (q_id, url_id) url_id
    FROM query_url
) q
GROUP BY url_id
ORDER BY c DESC
LIMIT 100
```

Natomiast Zapytanie 2 (tym razem zwracające w pełni oczekiwane wyniki) wykonywało się przez ok. 35 godzin, po czym zostało przerwane.

Listing 2: Zapytanie 2

```
SELECT url_id, COUNT(*) AS c
FROM(
    SELECT DISTINCT ON (q.query_id, u.url_id) u.url_id
    FROM query_url u
```

```
    INNER JOIN {0}.query q ON u.q_id=q.q_id
) q
GROUP BY url_id
ORDER BY c DESC
LIMIT 100
```

Wtedy postanowiono wrócić do oryginalnego pliku z danymi (lecz w postaci binarnej zamiast tekstowej) by wydobyć dane bezpośrednio z niego. Wymagało to czterokrotnego przejrzenia całego pliku - po razie na zliczenie zapytań dla każdego urla i termu, a następnie po kolejnym razie dla pobrania query_id dla każdego z najlepszych termów i urli (teoretycznie można by zmieścić się w dwóch przejrzeniach, jednak problemem okazała się zbyt mała ilość pamięci ram, aby pomieścić obszerne mapowania id na ilośc wystąpień i zbiór przeanalizowanych zapytań). Dopiero na etapie pisania tego programu udało się dojść do unikalności query_id (co było powodem zmiany schematu bazy danych), co pozwoliłoby na uzyskanie oczekiwanych wyników za pomocą zapytania zbliżonego do Zapytania 1, jednak nawet biorąc poprawkę na 3,5 krotne zmniejszenie objętości tabeli, to i tak samo znalezienie 100 najlepszych urli zajęłoby prawie 20 minut, a pobieranie listy zapytań dla każdego z urli trwałoby 2-3 minuty, co po dodaniu analogicznych operacji dla termów złożyłoby się na minimum 7-8 godzin, a do tego był już gotowy program, który zwracał dokładnie te same wyniki, ale w czasie 7-8 minut. Na wyjściu uzyskano plik składający się z listy urli/termów i odpowiadającej każdemu z nich listy zapytań.

Samo wyszukiwanie sąsiadów polegało na wczytaniu całego pliku, przetworzeniu go (w celu uzyskania listy urli/termów dla zapytania), a nastepnie porównywaniu kolejnych zapytań. Samo porównywanie, będące najbardziej czasochłonną częścią, udało się zrównoleglić, znacznie skracając czas przetwarzania.

Ze względu na stosunkowo wolny dostęp do danych zgromadzonych w utworzonym schemacie, dla problemu grupowania postanowiliśmy utworzyć osobną tabelę **Log**, zawierającą 201 kolumn:

- **query_id** czyli identyfikator zapytania
- **termN** (N $\in$ <1, 100>) 100 kolumn typu boolean, informujących czy dany term wystąpił w zapytaniu
- **urlN** (N $\in$ <1, 100>) 100 kolumn typu boolean, informujących czy dany url wystąpił w wynikach zapytania

do której dane importowaliśmy z użyciem list urli/termów dla każdego zapytania (uzyskanych identyczną metodą, jak w przypadku szukania najbliższych sąsiadów).

Samo grupowanie polegało na wykonywaniu zapytań postaci:

Listing 3: Zapytanie 3

```
SELECT COUNT(*), <Lista kolumn>
FROM log
GROUP BY <Lista kolumn>
```

gdzie <Lista kolumn> to albo nazwa pojedynczej kolumny (np. "url1") albo nazwa dwóch kolumn oddzielonych przecinkiem (np. "url1, url100").
Dodatkowo okazało się, że wąskim gardłem przy wykonywaniu zapytania nie był odczyt z dysku (co miało miejsce w przypadku dwóch poprzednich zapytań), tylko przetwarzanie na procesorze, dlatego również i to zadanie udało się zrównoleglić.

4 Eksperyment

Na czas wyszukiwania najbliższych sąsiadów składa się czas przetwarzania dwóch programów:

- wydobywającego top100 urli i termów i powiązanych z nimi zapytań - 469s
  - top100 urli - 133s
  - top100 termów - 83s
  - zapytania związane z urlami - 145s
  - zapytania związane z termami - 107s
- przeliczającego podobieństwa między zapytaniami - 264s (71s po zrównolegleniu)

Niestety otrzymane dane nie okazały się zbyt satysfakcjonujące (pierwsza kolumna to współczynnik Jaccarda, druga to liczba wystąpień danej wartości w 50 najbardziej zbliżonych zapytaniach).

similarity	count
1	4680
0,8571429	106
0,8	169
0,6666667	45

Table 2: Liczności prawdopodobieństw

Dodatkowo ze 100 przetworzonych zapytań udało się znaleźć jedynie 13, w których wystąpiła wartość różna od 1. Sprawia to, że wyniki nie są zbyt miarodajne i ciężko będzie na ich podstawie wydobyć jakieś wnioski.

Grupowanie wykorzystywało listę top100 urli i termów i powiązanych z nimi zapytań, uzyskanych na potrzeby szukania najbliższych sąsiadów, w dalszej kolejności należało dane wrzucić do bazy danych i wykonać odpowiednie zapytania. Import danych do bazy trwał 1h 23min 22s.

Czasy wykonywania zapytań prezentowały się następująco:

	po jednej kolumnie	po dwóch kolumnach
średni czas	16,937s	21,382s
odchylenie	9,010s	7,471s
czas minimalny	5,918s	6,508s
czas maksymalny	29,153s	29,095s

Table 3: Czasy wykonywania zapytań

przy czym warto zaznaczyć, że czas zależał od pozycji kolumny (im dalej się znajdowała, tym zapytanie trwało dłużej).

Otrzymane wyniki prezentują się następująco:

	po jednej kolumnie	po dwóch kolumnach
średnia liczność grupy	5601715	2800861 (dla niezerowych 3014466)
odchylenie	7792784	6636801 (dla niezerowych 6791040)
minimalna (niezerowa) liczność grupy	4354	1
maksymalna liczność grupy	11199074	11196049

Table 4: Grupowanie

5 Podsumowanie

Analiza danych sprawiła wiele trudności, jednak ostatecznie udało się ją przeprowadzić. Szablonowe podejście do problemu (wyszukiwanie sąsiadów) może się nie sprawdzić jako rozwiązanie dla tego zadania, ze względu na małą różnicę podobieństw pomiędzy poszczególnymi zapytaniami.

W obliczu zdobytych doświadczeń nadal nie jesteśmy jeszcze w stanie określić w jaki sposób przetwarzać te dane, dopuszczamy jednak możliwość, że czytanie bezpośrednio z wejściowych plików, okaże się dla nas najbardziej efektywne (nawet po uwzględnieniu wyższych kosztów implementacji).