O'REILLY®

Zwinna analiza danych Apache Hadoop dla każdego

ZARZĄDZAJ DUŻYMI ZBIORAMI DANYCH!

HELION

Russell Jurney

Tytuł oryginału: Agile Data Science: Building Data Analytics Applications with Hadoop

Tłumaczenie: Przemysław Szeremiota

ISBN: 978-83-246-9944-5

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog siążek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/zwiand.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/zwiand
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 2.

Dane

W tym rozdziale zapoznamy się ze zbiorem danych, na którym będziemy pracować w dalszej części książki; będzie to mianowicie nasza własna skrzynka odbiorcza poczty elektronicznej. Zapoznamy się też z szeregiem wykorzystywanych później narzędzi i przedstawimy uzasadnienie dla takiego, a nie innego ich wyboru. Na koniec zakreślimy perspektywy, które będziemy stosować w analizie danych.

Książka zaczyna się od danych, bo w zwinnym Big Data nasz proces też zaczyna się od danych.

Kto nie posiada konta pocztowego Gmail (*http://mail.google.com*), powinien je na potrzeby ćwiczeń założyć i zapełnić przykładowymi wiadomościami.

E-mail

Poczta elektroniczna, albo po prostu e-mail, to podstawowa usługa internetu. Jest nawet czymś więcej: powszechnym i podstawowym obecnie sposobem uwierzytelniania się w sieciach społecznościowych i najróżniejszych aplikacjach WWW. E-mail, mimo powszechności użycia, jest usługą złożoną, sygnał zawarty w danych poczty jest bogaty i pozwala na skuteczną eksplorację rozmaitych cennych informacji.

Jako zbiór danych dla aplikacji rozwijanej w toku kolejnych rozdziałów postanowiliśmy wybrać osobistą skrzynkę pocztową, ponieważ w ten sposób dane stają się istotne i cenne dla czytelnika. Pobierając zawartość skrzynki odbiorczej Gmail i używając jej w przykładach, natychmiast stajemy przed

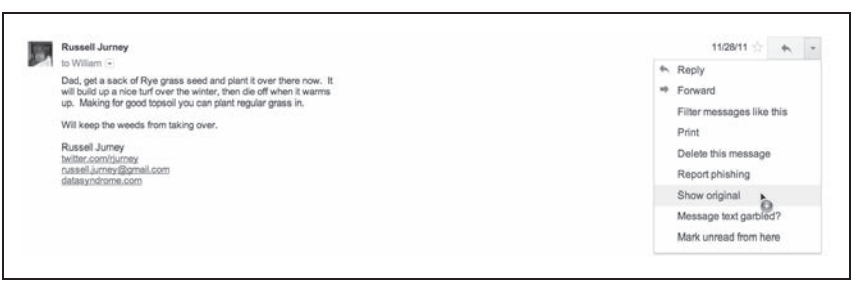
„dużym" (chociaż właściwie to „średnim") problemem przetwarzania danych: przetwarzanie danych wiadomości e-mail na własnym komputerze ledwo daje się przeprowadzić. Dane są przeważnie za duże, żeby zmieścić się w całości w pamięci, więc wymuszają stosowanie skalowalnych narzędzi (co pasuje do ducha zwinnego Big Data). Dalej, ponieważ będzie to nasza własna skrzynka, zyskamy istotne informacje ze świata bezpośrednio nas dotyczącego, co pozwoli na uwypuklenie wartości tkwiącej w tego rodzaju danych!

W książce będziemy odwoływać się do tych samych narzędzi, których użylibyśmy do przetwarzania danych w skali petabajtowej, tyle że będziemy je uruchamiać na własnym komputerze w trybie pracy lokalnej. W ten sposób nie tylko zapewnimy sobie efektywne przetwarzanie danych, ale też spełnimy postulat jednokrotnej budowy rozwiązania gotowego do użycia na większą skalę. Zwinność wymaga, aby wszystko, co robimy, było też możliwie proste.

Praca z surowymi danymi

Surowe wiadomości e-mail

Format wiadomości poczty elektronicznej jest rygorystycznie zdefiniowany w dokumencie RFC 5322 (*Request For Comments*, dokumenty stowarzyszenia Internet Engineering Task Force). Aby obejrzeć surową postać wiadomości e-mail w serwisie Gmail, należy wybrać wiadomość i z menu znajdującego się w prawym górnym narożniku wybrać opcję „Pokaż oryginał" (rysunek 2.1).



Rysunek 2.1. Opcja „Pokaż oryginał" w Gmailu

Surowa wiadomość e-mail wygląda tak:

```
From: Russell Jurney <russell.jurney@gmail.com>
Mime-Version: 1.0 (1.0)
```

```
Date: Mon, 28 Nov 2011 14:57:38 -0800
Delivered-To: russell.jurney@gmail.com
Message-ID: <4484555894252760987@unknownmsgid>
Subject: Re: Lawn
To: William Jurney <******@hotmail.com>
Content-Type: text/plain; charset=ISO-8859-1

Dad, get a sack of Rye grass seed and plant it over there now. It
will build up a nice turf over the winter, then die off when it warms
up. Making for good topsoil you can plant regular grass in.

Will keep the weeds from taking over.

Russell Jurney datasyndrome.com
```

Taką postać danych nazwiemy za pomocą określenia *dane na wpół ustrukturyzowane* (ang. *semi-structured data*).

Dane ustrukturyzowane a dane na wpół ustrukturyzowane

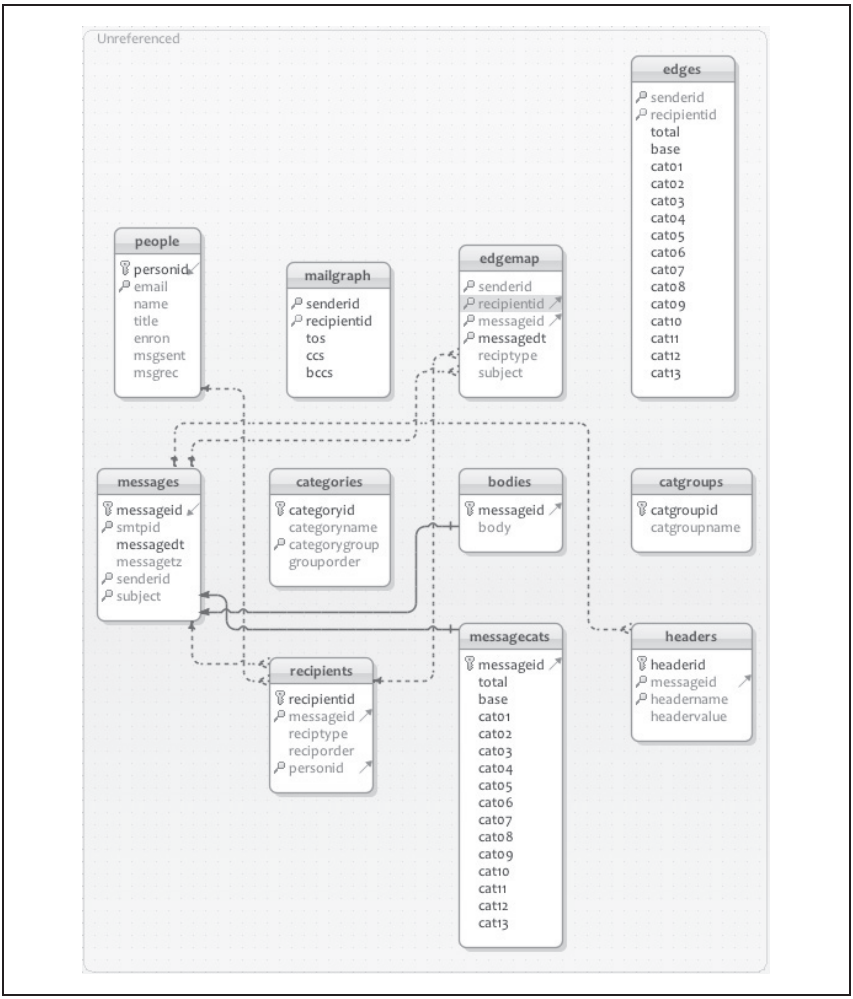
Wikipedia definiuje dane na wpół ustrukturyzowane jako:

> Formę danych ustrukturyzowanych, która nie spełnia formalnej struktury tabeli i modeli typowych dla relacyjnych baz danych, ale mimo to zawiera etykiety albo inne znaczniki oddzielające elementy semantyczne i narzucające hierarchię rekordów i pól wyodrębnionych w danych.

Taka definicja stoi w jawnej sprzeczności z danymi ustrukturyzowanymi, w których informacja jest wtłoczona w ściśle zdefiniowane schematy, jeszcze zanim zaczniemy jej analizę, a schemat jest optymalizowany do późniejszego wyłuskiwania danych. Ustrukturyzowaną perspektywę wiadomości e-mail dobrze ilustruje zbiór danych *Berkeley Enron* autorstwa Andrew Fiorego i Jeffa Heera, widoczny na rysunku 2.2.

SQL

Do odpytywania ustrukturyzowanych danych w schematach relacyjnych tradycyjnie wykorzystujemy deklaratywne języki programowania, takie jak SQL. W języku SQL określamy raczej to, co chcemy zrobić, a nie jak to zrobić. To zasadnicza różnica wobec programowania imperatywnego. W SQL-u określamy pożądany wynik operacji, a nie algorytm określający przebieg operacji. Zapytanie SQL wykonane wobec zbioru relacyjnego Enron w celu wydobycia pojedynczej, kompletnej wiadomości e-mail wyglądałoby tak:



Rysunek 2.2. Schemat wiadomości poczty elektronicznej Berkeley Enron

```
select m.smtpid as id,
    m.messagedt as date,
    s.email as sender,
    (select GROUP_CONCAT(CONCAT(r.reciptype, ':', p.email) SEPARATOR ' ')
        from recipients r
        join people p
            on r.personid=p.personid
        where r.messageid = 511) as to_cc_bcc,
    m.subject as subject,
```

```
    SUBSTR(b.body, 1, 200) as body
    from messages m
    join people s
        on m.senderid=s.personid
    join bodies b
        on m.messageid=b.messageid
    where m.messageid=511;

| <25772535.1075839951307.JavaMail.evans@thyme> | 2002-02-02 12:56:33
| pete.davis@enron.com |
to:pete.davis@enron.com cc:albert.meyers@enron.com cc:bill.williams@enron.com
cc:craig.dean@enron.com
cc:geir.solberg@enron.com cc:john.anderson@enron.com cc:mark.guzman@enron.com
cc:michael.mier@enron.com
cc:pete.davis@enron.com cc:ryan.slinger@enron.com bcc:albert.meyers@enron.com
bcc:bill.williams@enron.com
bcc:craig.dean@enron.com bcc:geir.solberg@enron.com bcc:john.anderson@enron.com
bcc:mark.guzman@enron.com
bcc:michael.mier@enron.com bcc:pete.davis@enron.com bcc:ryan.slinger@enron.com
| Schedule Crawler:
HourAhead Failure | Start Date: 2/2/02; HourAhead hour: 11;
HourAhead schedule download failed. Manual intervention required. |
```

Zauważmy, jak bardzo skomplikowane jest zapytanie wyłuskujące z tak ustrukturyzowanych danych pojedynczy, podstawowy rekord. Musimy łączyć różne tabele i używać podzapytań, specjalnej funkcji MySQL-a o nazwie `GROUP_CONCAT`, a także trochę bardziej typowych funkcji — `CONCAT` i `SUBSTR`.

Dane relacyjne niemal zawsze zniechęcają do przyjmowania pierwotnej perspektywy danych, przymuszając do postrzegania danych w ujęciu relacyjnym, a nie w ujęciu samej zawartości pierwotnych danych, które mają postać zupełnie zdenormalizowaną. Złożoność ta wpływa na każdą analizę, automatycznie przenosząc nas do „krainy SQL" zamiast do rzeczywistości konkretnych dokumentów.

Zauważmy też, że samo zdefiniowanie schematu również nie jest trywialne:

```
CREATE TABLE bodies (
    messageid int(10) unsigned NOT NULL default '0',
    body text,
    PRIMARY KEY (messageid)
) TYPE=MyISAM;

CREATE TABLE categories (
    categoryid int(10) unsigned NOT NULL auto_increment,
    categoryname varchar(255) default NULL,
    categorygroup int(10) unsigned default NULL,
    grouporder int(10) unsigned default NULL,
    PRIMARY KEY (categoryid),
    KEY categories_categorygroup (categorygroup)
```

```
) TYPE=MyISAM;

CREATE TABLE catgroups (
    catgroupid int(10) unsigned NOT NULL default '0',
    catgroupname varchar(255) default NULL,
    PRIMARY KEY (catgroupid)
) TYPE=MyISAM;

CREATE TABLE edgemap (
    senderid int(10) unsigned default NULL,
    recipientid int(10) unsigned default NULL,
    messageid int(10) unsigned default NULL,
    messagedt timestamp(14) NOT NULL,
    reciptype enum('bcc','cc','to') default NULL,
    subject varchar(255) default NULL,
    KEY senderid (senderid,recipientid),
    KEY messageid (messageid),
    KEY messagedt (messagedt),
    KEY senderid_2 (senderid),
    KEY recipientid (recipientid)
) TYPE=MyISAM;

CREATE TABLE edges (
    senderid int(10) unsigned default NULL,
    recipientid int(10) unsigned default NULL,
    total int(10) unsigned NOT NULL default '0',
    base int(10) unsigned NOT NULL default '0',
    cat01 int(10) unsigned NOT NULL default '0',
    cat02 int(10) unsigned NOT NULL default '0',
    cat03 int(10) unsigned NOT NULL default '0',
    cat04 int(10) unsigned NOT NULL default '0',
    cat05 int(10) unsigned NOT NULL default '0',
    cat06 int(10) unsigned NOT NULL default '0',
    cat07 int(10) unsigned NOT NULL default '0',
    cat08 int(10) unsigned NOT NULL default '0',
    cat09 int(10) unsigned NOT NULL default '0',
    cat10 int(10) unsigned NOT NULL default '0',
    cat11 int(10) unsigned NOT NULL default '0',
    cat12 int(10) unsigned NOT NULL default '0',
    cat13 int(10) unsigned NOT NULL default '0',
    UNIQUE KEY senderid (senderid,recipientid)
) TYPE=MyISAM;

CREATE TABLE headers (
    headerid int(10) unsigned NOT NULL auto_increment,
    messageid int(10) unsigned default NULL,
    headername varchar(255) default NULL,
    headervalue text,
    PRIMARY KEY (headerid),
    KEY headers_headername (headername),
    KEY headers_messageid (messageid)
) TYPE=MyISAM;
```

```
CREATE TABLE messages (
    messageid int(10) unsigned NOT NULL auto_increment,
    smtpid varchar(255) default NULL,
    messagedt timestamp(14) NOT NULL,
    messagetz varchar(20) default NULL,
    senderid int(10) unsigned default NULL,
    subject varchar(255) default NULL,
    PRIMARY KEY (messageid),
    UNIQUE KEY smtpid (smtpid),
    KEY messages_senderid (senderid),
    KEY messages_subject (subject)
) TYPE=MyISAM;

CREATE TABLE people (
    personid int(10) unsigned NOT NULL auto_increment,
    email varchar(255) default NULL,
    name varchar(255) default NULL,
    title varchar(255) default NULL,
    enron tinyint(3) unsigned default NULL,
    msgsent int(10) unsigned default NULL,
    msgrec int(10) unsigned default NULL,
    PRIMARY KEY (personid),
    UNIQUE KEY email (email)
) TYPE=MyISAM;
--
--

CREATE TABLE recipients (
    recipientid int(10) unsigned NOT NULL auto_increment,
    messageid int(10) unsigned default NULL,
    reciptype enum('bcc','cc','to') default NULL,
    reciporder int(10) unsigned default NULL,
    personid int(10) unsigned default NULL,
    PRIMARY KEY (recipientid),
    KEY messageid (messageid)
) TYPE=MyISAM;
```

Dla porównania — w zwinnym Big Data do definiowania postaci danych korzystamy z języków przepływu danych, a potem publikujemy dane wprost w bazie dokumentowej, nawet bez uprzedniego określania schematu! To podejście optymalizowane dla naszego procesu, czyli dla analizy danych, w której to analizie naszym zadaniem jest wyciągnięcie na podstawie posiadanych danych nowych informacji. Zewnętrzne narzucanie schematu danych nie daje tu żadnych korzyści i stanowi czysty narzut. W końcu przecież nie wiemy nawet, jak dokładnie będziemy z tych danych korzystać! Badania danych zawsze przynoszą niespodzianki.

Mimo to relacyjna struktura ma swoje zalety; możemy na przykład łatwo sprawdzić, w których godzinach użytkownik wysyła wiadomości, za pomocą prostego zapytania grupującego i sortującego:

```
select senderid as id,
    hour(messagedt) as sent_hour,
    count(*)
    from messages
    where senderid=511
    group by
        senderid,
        m_hour
    order by
        senderid,
        m_hour;
```

Zapytanie to zwróci poniższą prostą tabelkę:

```
+----------+--------+----------+
| senderid | m_hour | count(*) |
+----------+--------+----------+
|        1 |      0 |        4 |
|        1 |      1 |        3 |
|        1 |      3 |        2 |
|        1 |      5 |        1 |
|        1 |      8 |        3 |
|        1 |      9 |        1 |
|        1 |     10 |        5 |
|        1 |     11 |        2 |
|        1 |     12 |        2 |
|        1 |     14 |        1 |
|        1 |     15 |        5 |
|        1 |     16 |        4 |
|        1 |     17 |        1 |
|        1 |     19 |        1 |
|        1 |     20 |        1 |
|        1 |     21 |        1 |
|        1 |     22 |        1 |
|        1 |     23 |        1 |
+----------+--------+----------+
```

Relacyjne bazy danych dzielą dane na tabele, zgodnie z określoną strukturą relacji, i wstępnie obliczają indeksy potrzebne do łączenia danych z tych tabel. Indeksy pozwalają tym systemom osiągnąć dużą szybkość odpowiedzi z wykorzystaniem pojedynczej maszyny. Do odpytywania o dane w takiej strukturze stosuje się programowanie deklaratywne.

Tego rodzaju programowanie świetnie nadaje się do konsumowania i przeszukiwania ustrukturyzowanych danych w agregatach, dzięki czemu można łatwo generować proste wykresy i diagramy. Kiedy już dokładnie wiadomo, co jest w danych i co nas z tego interesuje, możemy efektywnie wysterować mechanizm SQL, który za nas wyznaczy i pobierze odpowiednie zbiory danych łączonych w relacje. Nie musimy zajmować się szczegółami samego sposobu wykonania zapytania.

NoSQL

Inaczej niż w SQL-u, kiedy budujemy aplikacje analityczne, często nie wiemy jeszcze, jakie zapytania będziemy w nich wykonywać, to znaczy, jakie przekroje i agregaty danych będą nas interesować. Wypracowanie rozwiązania wymaga wielu iteracji implementacji i eksperymentów. Dane są często niedostępne w formacie relacyjnym: dane przenikające świat rzeczywisty rzadko są znormalizowane, raczej zaszumione i rozmyte. Samo wyodrębnienie ich struktury bywa długotrwałym procesem, który wykonujemy iteracyjnie, w miarę dostrzegania i wyłuskiwania kolejnych cech danych.

Z tych powodów w zwinnym Big Data stosujemy przede wszystkim programowanie imperatywne, implementowane w systemach rozproszonych. Języki imperatywne, takie jak Pig Latin, opisują czynności potrzebne do manipulowania danymi przetwarzanymi potokowo. Zamiast obliczać indeksy rekordów posiadających ustaloną strukturę, do wyłuskiwania pojedynczych rekordów używamy wielu współbieżnie działających procesów. Cały ten przepływ danych jest możliwy dzięki narzędziom takim jak Hadoop i kolejki zadań (ang. *work queue*).

Języki imperatywne nie tylko dobrze przekładają się na model pracy w technologiach w rodzaju Hadoop (co pozwala na łatwe skalowanie przetwarzania), ale też kładą nacisk na te obszary aplikacji analitycznych, w których odbywa się najwięcej przetwarzania: będzie to jedna czy dwie kluczowe operacje, w których będziemy wytwarzać gros wartości naszej aplikacji.

Wypracowanie tych najbardziej cennych operacji przetwarzających jest — w porównaniu do pisania zapytań SQL — procesem długotrwałym i często żmudnym, ponieważ nasze przetwarzanie może angażować techniki z dziedzin statystyki, uczenia maszynowego czy analizy sieci społecznych. Programowanie imperatywne jest wtedy jedyną możliwością.

Podsumowując, kiedy schematy są ściśle określone, a do dyspozycji mamy tylko SQL, nasze postrzeganie danych jest zdominowane przez dostępność narzędzi optymalizowanych do konsumpcji gotowych relacji, a nie do wyszukiwania nowych przez eksplorację surowych danych. Sztywne schematy są wtedy przeszkodą, blokując możliwość intuicyjnego łączenia aspektów danych. Z kolei praca z danymi nieustrukturyzowanymi pozwala na skupienie się wprost na danych, iteracyjne wyłuskiwanie wartości i przekształcanie ich w produkt. W Big Data, jeśli chcemy zachować zwinność, użyjemy technik NoSQL.

Serializacja

Z naszymi danymi wpółustrukturyzowanymi moglibyśmy pracować wprost, operując na czystym tekście, ale pomocne będzie narzucenie im chociaż najluźniejszej z możliwych struktury. Możliwość tę dają nam systemy serializacji danych, takie jak:

Thrift: *http://thrift.apache.org/*

Protobuf: *http://code.google.com/p/protobuf/*

Avro: *http://avro.apache.org/*

Na potrzeby omówienia wybierzemy Avro, mimo że jest to mechanizm najmniej dojrzały i uznany z wymienionych. Avro pozwala na serializowanie nawet złożonych struktur danych, do każdego pliku zserializowanych danych dołącza ich schemat, a ponadto jest obsługiwany w Apache Pig.

Instalacja Avro jest prosta, a samo oprogramowanie nie wymaga odwoływania się do zewnętrznych usług.

Zdefiniujemy sobie teraz prosty schemat Avro odpowiedni dla wiadomości e-mail zgodnych z definicją z RFC 5322. Dobrze jest mieć zawczasu ustalony schemat, choć w praktyce zwinnego Big Data większość operacji przetwarzających i tak będzie musiała wyodrębniać i przetwarzać wszystkie encje tego schematu. Dlatego pierwotny schemat może równie dobrze być mocno uproszczony, jak tutaj:

```
{
    "type":"record",
    "name":"RawEmail",
    "fields":
    [
        {
            "name":"thread_id",
            "type":["string", "null"],
            "doc":""
        },
        {
            "name":"raw_email",
            "type": ["string", "null"]
        }
    ]
}
```

Schemat moglibyśmy uprościć jeszcze bardziej: wyodrębnić jedynie `thread_id` jako unikatowy identyfikator wiadomości, a resztę jej treści umieścić w pojedynczym polu surowych danych. Jeśli z surowych rekordów trudno wyodrębnić jednoznaczny i unikatowy identyfikator rekordów, możemy

zewnętrznie nadawać rekordom identyfikatory UUID (*Universally Unique IDentifier*) i dokładać je w osobnym polu.

W toku przetwarzania danych naszym zadaniem jest dodawanie pól do schematu w miarę wyodrębniania kolejnych interesujących aspektów poszczególnych rekordów, przy jednoczesnym zachowaniu surowych danych źródłowych w osobnym polu (jeśli tylko możemy sobie na to pozwolić); wtedy w kolejnych etapach przetwarzania wciąż będzie można odwołać się do nieprzetworzonego źródła danych.

Wyodrębnianie i ujawnianie cech w ewoluującym schemacie

Pete Warden w swojej prezentacji „Embracing the Chaos of Data" (*http://bit.ly/171lulz7*) zauważył, że większość dostępnych dla nas danych to dane nieustrukturyzowane i nieoczyszczone. Dostępność olbrzymich ilości takich właśnie danych, nieprzygotowanych nijak do umieszczania w znormalizowanych tabelach, to synonim i sedno „big data". To równocześnie szansa wyekstrahowania z surowych i nieoczyszczonych danych istotnych informacji i wykorzystania tych informacji jako bazy do kolejnych, nowych operacji.

Cechy wyodrębniane z nieustrukturyzowanych danych są oczyszczane wyłącznie w pełnym świetle dziennym, to znaczy, kiedy trafiają do konsumujących je użytkowników, a ci zgłaszają pretensje i uwagi; jeśli nie możemy publikować cech danych w miarę ich wyodrębniania, nasz produkt nie będzie się rozwijał. Najtrudniejszym elementem budowania produktów opartych na danych jest dodawanie ekstrakcji encji i cech do produktów jeszcze znacznie mniejszych niż w docelowej wizji. Dlatego schemat powinien rozpoczynać się jako wór z nieustrukturyzowanym tekstem i dopiero stopniowo ewoluować do danych ustrukturyzowanych, w miarę wyodrębniania istotnych cech.

Cechy te muszą być eksponowane w jakiejś formie produktowej, inaczej nigdy nie osiągną stanu gotowości produkcyjnej. Wypreparowane dane, które trzymamy w niedostępnych piwnicach naszego produktu, raczej nie mają szansy na docelowe ukształtowanie, bo nie ma takiej presji. Dlatego lepiej od razu wszystkie wyodrębniane cechy danych ujawniać na stronach encji w aplikacji, w postaci gotowej do konsumpcji i stopniowego ulepszania. Aplikacja może stopniowo uwidaczniać agregaty i kombinacje tych encji; efekt będzie na pewno lepszy niż w przypadku próby uwidocznienia wszystkich cech, które przychodzą nam do głowy jeszcze przed ich ujawnieniem

w danych (bo w danych zawarte są nieodzownie również takie informacje, których obecność nie była oczywista).

Podczas eksploracji danych do postaci ustrukturyzowanej informacji używamy tej informacji do ujawniania nowych faktów i stawiania prognoz; to sprzężenie stanowi gigantyczny potencjał w zakresie wytwarzania rzeczywistej wartości aplikacji. Dane są brutalne i nieobliczalne; zignorowanie ich prawdziwej, ujawniającej się stopniowo natury zniweczy marzenia nawet najbardziej ambitnych menedżerów produktu.

W książce przekonamy się niejednokrotnie, że ewolucja i ulepszanie schematu danych odbywa się wraz z ujawnianiem nowych cech tych danych. A prawdziwą zwinność osiągamy wtedy, kiedy ewolucja schematu odbywa się równolegle.

Potoki danych

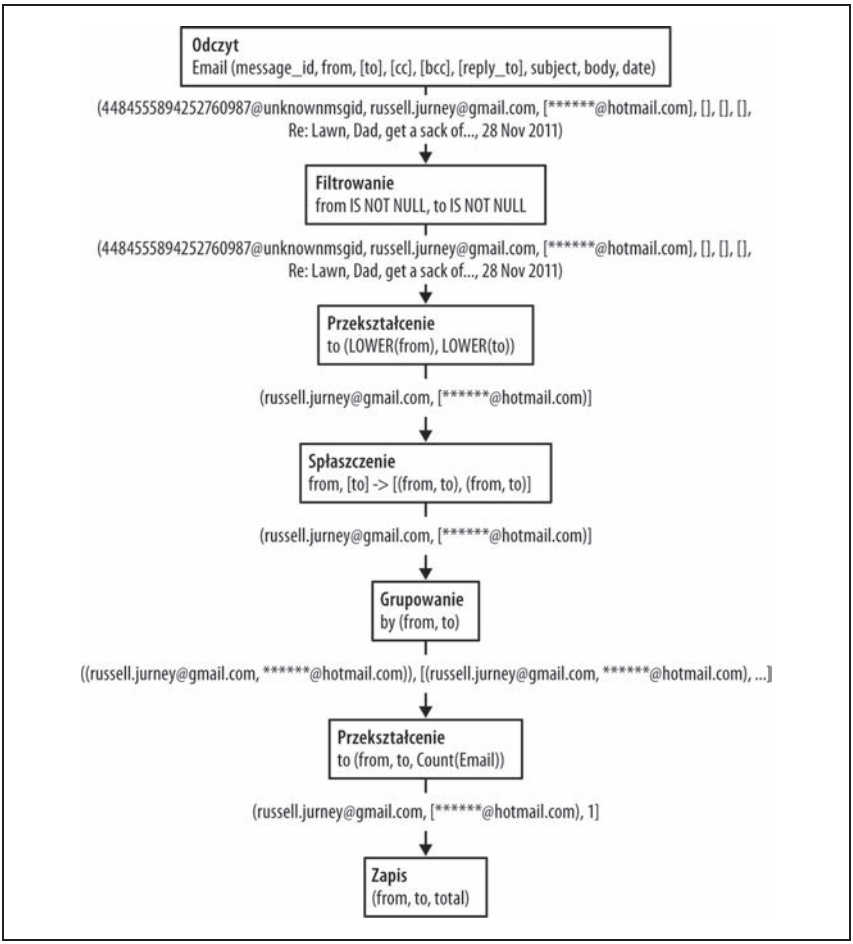
Będziemy pracować z na wpół ustrukturyzowanymi danymi przesyłanymi w potokach danych, w których będziemy wypreparowywać i prezentować poszczególne wykryte cechy danych. Zaletą takiego modelu pracy z danymi jest brak inwestycji czasowej w wyodrębnianie struktury dopóty, dopóki nie okaże się ona interesująca i przydatna. Zgodnie z zasadami KISS (*Keep It Simple, Stupid!*) i YAGNI (*You Ain't Gonna Need It*) opóźniamy ten narzut do momentu, w którym okaże się niezbędny. Dzięki wykorzystywanym narzędziom będzie to również podejście bardziej efektywne, o czym przekonamy się w rozdziale 3.

Na rysunku 2.3 widać schemat wieloetapowego potoku danych obliczającego liczbę wiadomości wysłanych pomiędzy dwoma adresami e-mail.

Na pierwszy rzut oka przepływ danych wydaje się skomplikowany, zwłaszcza w porównaniu z SQL-em, ale szybko przekonamy się, że ten model pracy jest prostszy, a przetwarzanie danych w potokach stanie się naszą drugą naturą.

Perspektywy danych

Na dobry początek warto zapoznać się z różnymi sposobami postrzegania danych o wiadomościach e-mail. W zwinnym Big Data stosujemy zróżnicowane perspektywy i wielorakie tryby eksploracji danych między innymi dlatego, że przyjęcie jednej perspektywy często kończy się wyczerpaniem kreatywności eksploracji danych i ograniczeniem się do jednego czy dwóch



Rysunek 2.3. Prosty przepływ danych ze zliczaniem liczby wiadomości przesłanych pomiędzy parą kont poczty elektronicznej

sposobów, które wcześniej okazały się skuteczne. Potem przyjrzymy się bliżej różnym perspektywom wiadomości e-mail — będziemy je stosować w dalszym omówieniu.

Sieci

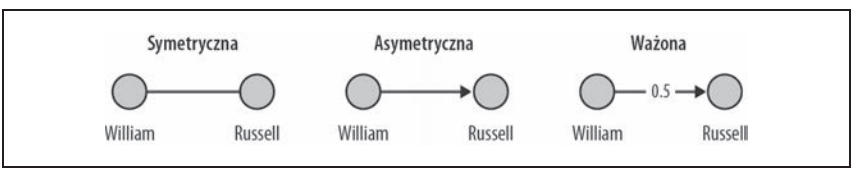
Sieć społeczna to grupa osób (ego) i połączenia pomiędzy nimi. Połączenia te mogą być skierowane, na przykład kiedy mówimy, że „Bob zna Alicję";

mogą być też nieskierowane: „Bob i Alicja są przyjaciółmi”. Połączenia mogą również mieć przypisane siły albo inaczej wagi: „Bob dobrze zna Alicję” (z wagą na przykład w skali od 1 do 10) czy też „Bob i Alicja są małżeństwem” (w skali od 0 do 1).

Do utworzenia sieci społecznej można użyć nadawców i odbiorców wiadomości e-mail z pól `from`, `to`, `cc` i `bcc`. Na przykład poniższa wiadomość definiuje dwie encje: `russell.jurney@gmail.com` oraz `******@hotmail.com`:

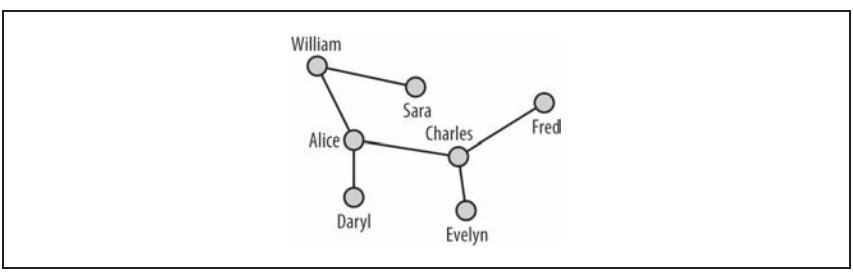
```
From: Russell Jurney <russell.jurney@gmail.com>
To: ******* Jurney <******@hotmail.com>
```

Wiadomość sama w sobie implikuje powiązanie pomiędzy tymi encjami. Możemy je wtedy reprezentować w prostej sieci społecznej, jak na rysunku 2.4.



Rysunek 2.4. Diady sieci społecznej

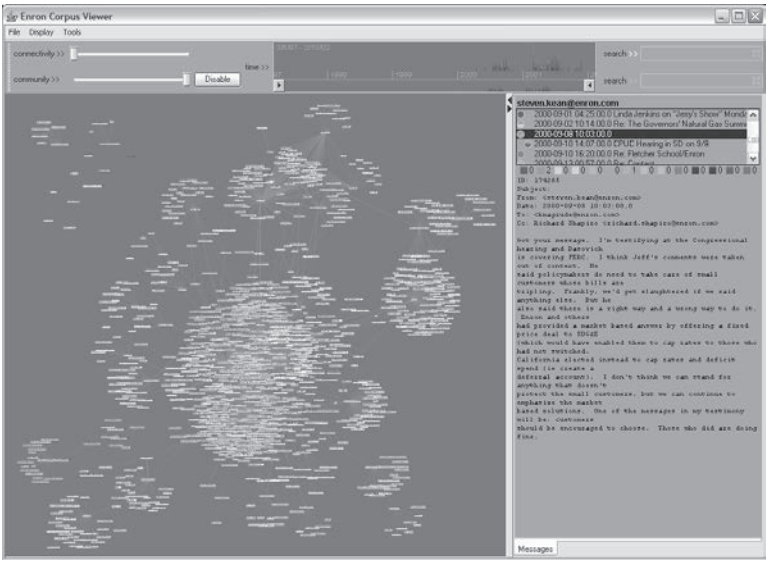
Na rysunku 2.5 mamy przykład nieco bardziej rozbudowanej sieci społecznej.



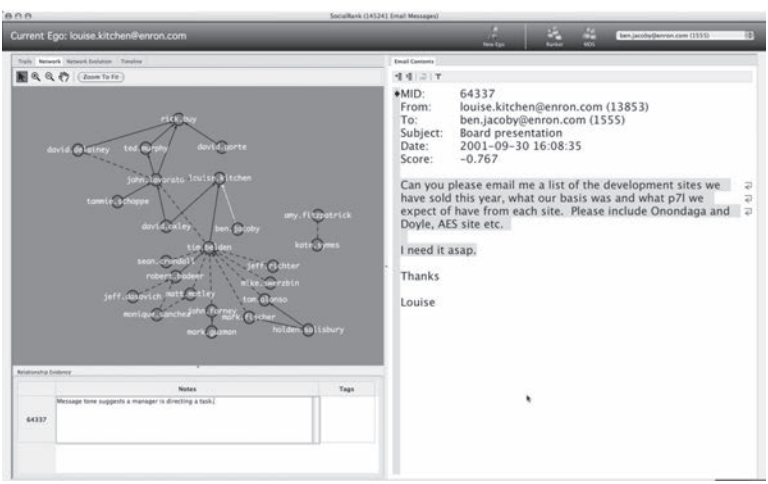
Rysunek 2.5. Sieć społeczna

Na rysunku 2.6 widać przykład sieci społecznej zbudowanej z 200-megabajtowej próbki wiadomości e-mail z Enrona.

Analiza sieci społecznych, w skrócie SNA (od *Social Network Analysis*), to metoda naukowego badania i analizy sieci społecznych. Modelując swoją skrzynkę odbiorczą jako sieć społeczną, możemy korzystać z metod SNA (np. z algorytmu PageRank) do wyekstrahowania głębszej wiedzy o danych i tym samym o naszej osobistej sieci społecznej. Tego rodzaju analizę, przeprowadzoną dla sieci Enron, widać na rysunku 2.7.



Rysunek 2.6. Sieć społeczna Enron wizualizowana w Enron Corpus Viewer (Jeffrey Heer i Andrew Fiore)

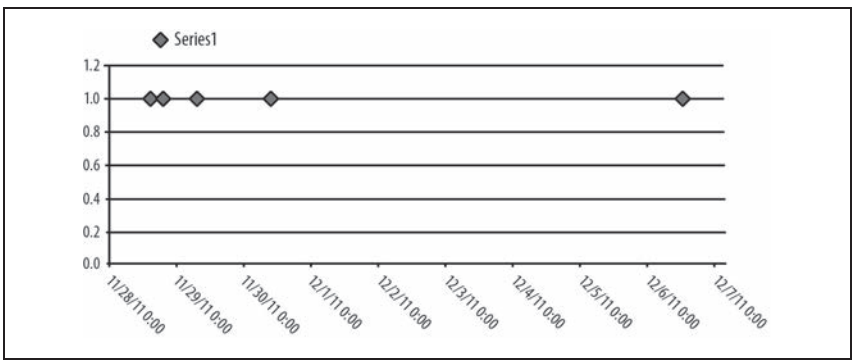


Rysunek 2.7. Enron SocialRank (Jaime Montemayor, Chris Diehl, Mike Pekala i David Patrone)

Szeregi czasowe

Szereg czasowy to sekwencja punktów danych uporządkowana według znacznika czasowego, zarejestrowanego wraz z każdą wartością. Szeregi czasowe pozwalają na obserwowanie zmian i trendów w danych, ujawniających się z upływem czasu. Wszystkie wiadomości e-mail mają znaczniki czasowe, co pozwala reprezentować poszczególne wiadomości w szeregach czasowych, jak pokazano na rysunku 2.8:

```
Date: Mon, 28 Nov 2011 14:57:38 -0800
```



Rysunek 2.8. Surowe szeregi czasowe

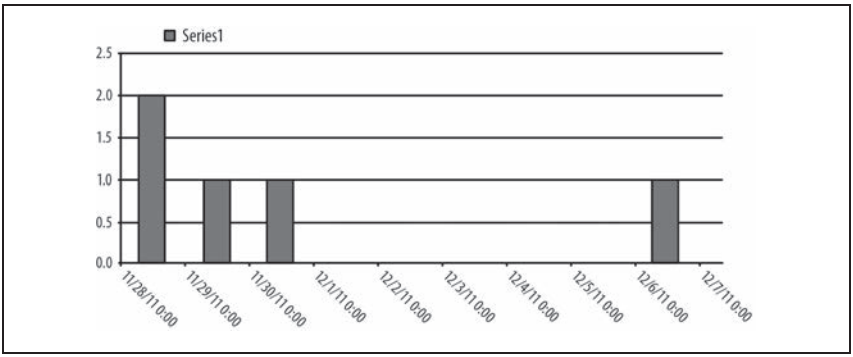
Dysponując większą próbką wiadomości, możemy skutecznie wyrysować surowe dane wiadomości w szeregach czasowych.

Ponieważ tutaj nie analizujemy dodatkowych wartości skojarzonych z szeregami czasowymi, możemy spróbować lepiej uwidocznić charakter danych przez pogrupowanie ich w odcinkach dobowych (patrz rysunek 2.9). Będziemy wtedy widzieć, jak wiele wiadomości zostało wysłanych pomiędzy parami adresów dziennie.

Analiza szeregów czasowych może ujawniać momenty szczególnego natężenia odbierania wiadomości e-mail od danych osób, a nawet pozwolić wywnioskować, jaki jest harmonogram czasu pracy tych osób.

Język naturalny

Sednem wiadomości poczty elektronicznej jest ich treść. Mimo powszechnego stosowania załączników multimedialnych poczta elektroniczna to wciąż przede wszystkim tekst:



Rysunek 2.9. Pogrupowane szeregi czasowe

```
Subject: Re: Lawn
Content-Type: text/plain; charset=ISO-8859-1

Dad, get a sack of Rye grass seed and plant it over there now. It
will build up a nice turf over the winter, then die off when it warms
up. Making for good topsoil you can plant regular grass in.

Will keep the weeds from taking over.

Russell Jurney
twitter.com/rjurney
russell.jurney@gmail.com
datasyndrome.com
```

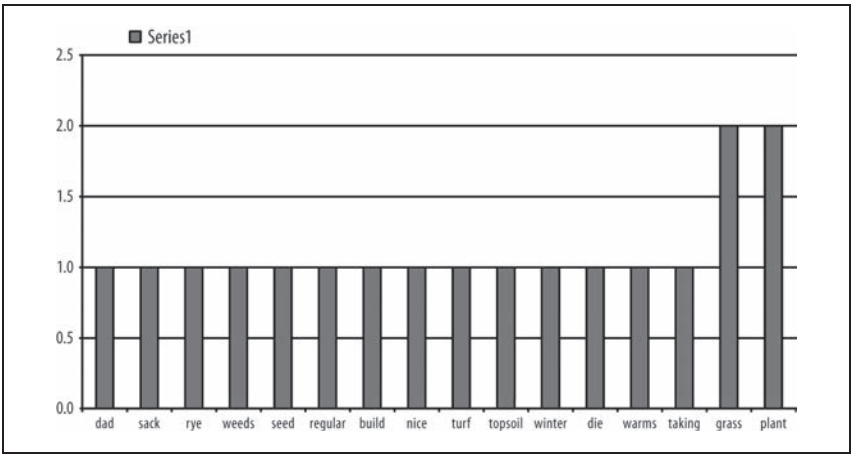
Treść wiadomości możemy analizować przez zliczanie częstości występowania poszczególnych słów. Po wyeliminowaniu nieznaczących słów typowych (tzw. *stopwords*, takich jak „to", „czy", „i") wygląda to tak jak na rysunku 2.10.

Częstość występowania słów możemy wykorzystać do wywnioskowania tematu wiadomości: ta konkretna wiadomość wydaje się być poświęcona roślinom i trawie, bo tego dotyczą najczęściej występujące słowa. Takie przetwarzanie języka naturalnego pozwala na wyodrębnienie cech nadających danym lepiej zdefiniowaną strukturę, co umożliwia włączenie tych ustrukturyzowanych cech do dalszej analizy.

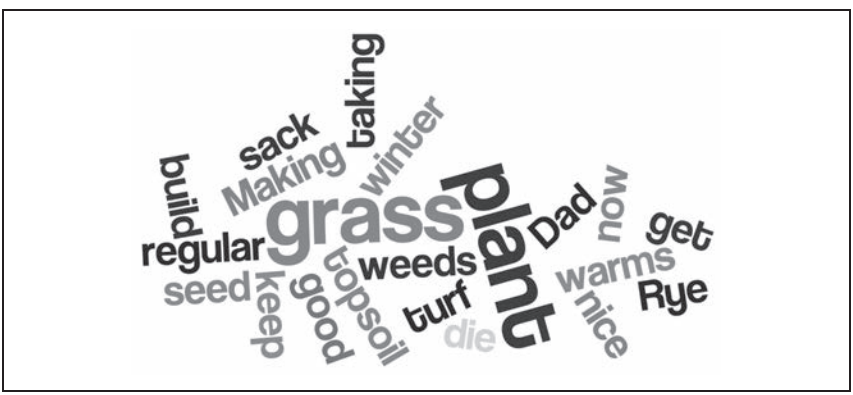
Zabawnym sposobem ilustrowania częstości występowania słów są chmury słów, takie jak na rysunku 2.11.

Prawdopodobieństwo

W teorii prawdopodobieństwa modelujemy proces pozornie losowy, zliczając wystąpienia i sąsiedztwo wystąpień poszczególnych właściwości, co



Rysunek 2.10. Częstość występowania słów w treści wiadomości



Rysunek 2.11. Chmura słów z treści wiadomości

pozwala na wygenerowanie rozkładów prawdopodobieństwa. Owe rozkłady możemy potem zaangażować do generowania sugestii i klasyfikowania encji do różnych kategorii.

Rozkłady prawdopodobieństwa możemy też wykorzystywać do stawiania prognoz. Moglibyśmy na przykład wygenerować rozkład prawdopodobieństwa dla wiadomości wysłanych, na podstawie zawartości pól `from`, `to`, i `cc`. Można wtedy próbować odpowiadać na pytania w rodzaju: jeśli nadawcą jest `russell.jurney@gmail.com`, a wiadomość jest skierowana na pewien konkretny adres, to jaka jest szansa, że w polu `cc` wystąpi inny konkretny adres poczty elektronicznej?

W tym przypadku nasze surowe dane to zawartość pól `from`, `to` i `cc` z poszczególnych wiadomości:

```
From: Russell Jurney <russell.jurney@gmail.com>
To: ****** Jurney <******@hotmail.com>
Cc: Ruth Jurney <****@hotmail.com>
```

Aby przeprowadzić naszą analizę, zliczamy pary adresów `from` i `to`, czyli współwystępowanie adresów poczty elektronicznej. Wyróżnijmy jedną parę: to wiadomości *wymieniane* pomiędzy mną a redaktorem wydawnictwa O'Reilly, Mikiem Loukidesem (tabela 2.1).

Tabela 2.1. Częstość współwystępowania adresów from-to

Od (from)	Do (to)	Liczba
russell.jurney@gmail.com	*****.jurney@gmail.com*	10
russell.jurney@gmail.com	*toolsreq@oreilly.com*	10
russell.jurney@gmail.com	*yoga*****@gmail.com*	11
russell.jurney@gmail.com	*user@pig.apache.org*	14
russell.jurney@gmail.com	******@hotmail.com*	15
russell.jurney@gmail.com	***mikel@oreilly.com***	**28**
russell.jurney@gmail.com	*russell.jurney@gmail.com*	44

Podzielenie tych wartości przez łączną liczbę wiadomości da w wyniku rozkład prawdopodobieństwa opisujący szansę, że dana wiadomość od takiego nadawcy będzie przeznaczona do danego adresata (tabela 2.2).

Tabela 2.2. Prawdopodobieństwo współwystępowania adresów from-to

Od (from)	Do (to)	Prawdopodobieństwo
russell.jurney@gmail.com	*****.jurney@gmail.com*	0,0359
russell.jurney@gmail.com	*toolsreq@oreilly.com*	0,0359
russell.jurney@gmail.com	*yoga*****@gmail.com*	0,0395
russell.jurney@gmail.com	*user@pig.apache.org*	0,0503
russell.jurney@gmail.com	******@hotmail.com*	0,0539
russell.jurney@gmail.com	***mikel@oreilly.com***	0,1007
russell.jurney@gmail.com	*russell.jurney@gmail.com*	0,1582

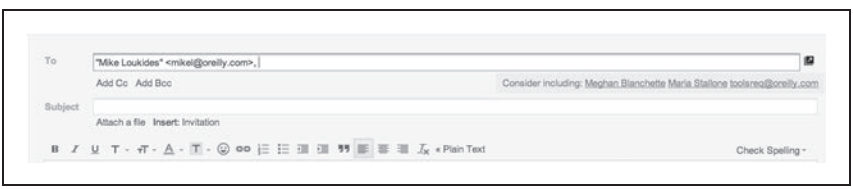
Na koniec obliczamy prawdopodobieństwo współwystępowania par odbiorców dla danego adresata wiadomości (tabela 2.3).

Dane te możemy wykorzystać do podpowiadania, kto jeszcze być może powinien być odbiorcą wiadomości, jeśli jej właściwym adresatem jest dana

Tabela 2.3. P(cc | from ∩ to): prawdopodobieństwo współwystąpienia cc dla wystąpienia pary nadawca-adresat

Nadawca (from)	Adresat (to)	Odbiorca (cc)	Prawdopodobieństwo
russell.jurney@gmail.com	***mikel@oreilly.com***	***toolsreq@oreilly.com***	**0.0357**
russell.jurney@gmail.com	***mikel@oreilly.com***	***meghan@oreilly.com***	**0,25**
russell.jurney@gmail.com	***mikel@oreilly.com***	***mstallone@oreilly.com***	**0,25**
russell.jurney@gmail.com	*toolsreq@oreilly.com*	*meghan@oreilly.com*	0,1
russell.jurney@gmail.com	*toolsreq@oreilly.com*	*mikel@oreilly.com*	0,2

osoba. Tego rodzaju dane można wykorzystywać do sterowania mechanizmami sugestii, takimi jak w podpowiadaniu odbiorców w poczcie Gmail (patrz rysunek 2.12).



Rysunek 2.12. Podpowiadanie odbiorców w poczcie Gmail

Później zobaczymy, jak za pomocą wnioskowania bayesowskiego generować użyteczne podpowiedzi odbiorców, nawet przy niepełnych danych prawdopodobieństwa współwystępowania (to znaczy, kiedy mamy luki w tabeli 2.3)

Podsumowanie

Przekonaliśmy się, że analizując wpółustrukturyzowane dane za pomocą różnych algorytmów i perspektyw, możemy uzyskać lepsze ukierunkowanie przyszłych zastosowań niż w przypadku ściśle ustrukturyzowanych i znormalizowanych zestawień relacyjnych. Zdefiniowane tu perspektywy będziemy stosować w kolejnych rozdziałach do wytwarzania funkcjonalności, w miarę wspinania się w górę piramidy wartości danych. A w następnym rozdziale nauczymy się określać schematy dla naszego składowiska danych za pomocą Apache Pig.

Skorowidz

R

S

Ś

T

U

W

Z

O'REILLY®

Zwinna analiza danych Apache Hadoop dla każdego

W dobie Big Data klasyczne podejście do analizy danych nie przynosi już pożądanych wyników. Skuteczna analiza gigantycznych zbiorów informacji, wyciąganie interesujących wniosków i prezentowanie ich w przejrzystej formie użytkownikowi wymagają mnóstwa czasu i środków. Zastanawiasz się, jak podejść do tego problemu, by zminimalizować ryzyko niepowodzenia? Na to i wiele innych pytań odpowiada ta książka.

Dzięki niej dowiesz się, jak wykorzystać platformę Hadoop do własnych celów. Skorzystasz z prostych narzędzi, takich jak język Python, biblioteka D3.js oraz Apache Pig, i zastosujesz zwinne podejście do problemu, by osiągnąć zaskakujące efekty. Ponadto przekonasz się, jak łatwo można publikować dane w MongoDB, stosować wyszukiwarkę Elasticsearch oraz wykorzystać potencjał chmur obliczeniowych. Nauczysz się także wizualizować dane na wykresach, prognozować oraz podejmować właściwe działania. Książka ta jest doskonałą lekturą dla wszystkich osób stojących przed problemem skutecznej pracy z ogromnymi zbiorami danych.

Dzięki tej książce:

- poznasz najlepsze narzędzia do przetwarzania zbiorów danych
- wykorzystasz możliwości języka Python
- sprawdzisz możliwości chmur obliczeniowych
- błyskawicznie wyszukasz dane za pomocą Elasticsearch
- zwizualizujesz dane z użyciem D3.js

Zwinnie rozwiąż problemy z dużymi zbiorami danych!

helion.pl księgarnia internetowa

Nr katalogowy: 28009

Księgarnia internetowa: http://helion.pl

Zamówienia telefoniczne: 0 801 339900 0 601 339900

Helion

Sprawdź najnowsze promocje: http://helion.pl/promocje
Książki najchętniej czytane: http://helion.pl/bestsellery
Zamów informacje o nowościach: http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-246-9944-5

Cena 39,90 zł

Informatyka w najlepszym wydaniu