Finding trending topics using Google Books n-grams data and Apache Hive on Elastic MapReduce

This article gives an introduction to processing n-gram data using Amazon Elastic MapReduce and Apache Hive. In this example we will calculate the top trending topics per decade. The data used to find the topics comes from the Google Book's n-gram corpus.

Details

Submitted By: Andrew Hitchcock

AWS Products Used: Elastic MapReduce

Created On: December 23, 2010 10:31 PM GMT

Last Updated: December 23, 2010 10:31 PM GMT

This article gives an introduction to processing n-gram data using Amazon Elastic MapReduce and Apache Hive. In this example we will calculate the top trending topics per decade. The data used to find the topics comes from the Google Book's n-gram corpus.

The n-gram data is licensed under a Creative Commons Attribution 3.0 Unported License. The original dataset is available from the Google Ngrams viewer website.

Amazon Elastic MapReduce is a web service which provides easy access to a Hadoop MapReduce cluster in the cloud. Apache Hive is an open source data warehouse built on top of Hadoop which provides a SQL-like interface over MapReduce.

In order to calculate the trending topics, we are going to have to go through a few steps. We'll launch an Elastic MapReduce job flow in which to do the processing. Then we'll set up Hive and the create the appropriate tables to load data from. Next we'll do some preprocessing to normalize the data by converting it to lower case and ignoring certain characters. We will aggregate the word counts per decade and find each word's usage fraction by decade. Finally, we'll do a decade by decade comparison to find the top words.

Starting a job flow with Hive

Before we can begin we must launch an Elastic MapReduce job flow with Hive installed. This will give us a SQL-like interface to the n-gram data and make it very easy to start processing it. We can launch a job flow using the Elastic MapReduce ruby client. If you've never used Elastic MapReduce before, you should read about how to install and configure the ruby client. That page also explains how to sign up for Elastic MapReduce if you are a new user. Once everything is set up, start a job flow with this command.

```
elastic-mapreduce --create --alive --hive-interactive --hive-versions 0.7
```

This will give you a job flow ID. You can track the status of your job low with the list parameter.

```
elastic-mapreduce --list <job-flow-id>
```

In a few minutes this will start and enter the waiting state, at which point we can SSH to the master node.

```
elastic-mapreduce --ssh <job-flow-id>
```

Now that we have Hive installed on our cluster we can log into its shell by typing 'hive'.

Public Data Sets
Release Notes
Sample Code & Libraries
Security Center
Videos & Webinars


```
$ hive
```

There are two settings that we need to set in order to efficiently process the data from Amazon S3. Type these two commands into the Hive shell.

```
hive> set hive.base.inputformat=org.apache.hadoop.hive.ql.io.HiveInputFormat;
hive> set mapred.min.split.size=134217728;
```

Since the data is stored in Amazon S3 in a single file, if we don't specify these settings we will only use one mapper when processing the data, which doesn't take advantage of the distributed nature of MapReduce. These commands tell Hive to use an InputFormat that will split a file into pieces for processing, and tell it not to split them into pieces any smaller than 128 MB.

Creating input table

In order to process any data, we have to first define the source of the data. We do this using a CREATE TABLE statement. For the purpose of this example, I only care about English 1-grams. This is the statement I use to define that table.

```
CREATE EXTERNAL TABLE english_1grams (
 gram string,
 year int,
 occurrences bigint,
 pages bigint,
 books bigint
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS SEQUENCEFILE
LOCATION 's3://datasets.elasticmapreduce/ngrams/books/20090715/eng-all/1gram/';
```

This statement creates an external table (one not managed by Hive) and defines five columns representing the fields in the dataset. We tell Hive that these fields are delimited by tabs and stored in a SequenceFile. Finally we tell it the location of the table.

Normalizing the data

The data in its current form is very raw. It contains punctuation, numbers (typically years), and is case sensitive. For our use case we don't care about most of these and only want clean data that is nice to display. It is probably enough to filter out n-grams with certain non-alphabetical characters and then to lowercase everything.

First we want to create a table to store the results of the normalization. In the process, we can also drop unnecessary columns.

```
CREATE TABLE normalized (
 gram string,
 year int,
 occurrences bigint
);
```

Now we need to insert data into this table using a select query. We'll read from the raw data table and insert into this new table.

```
INSERT OVERWRITE TABLE normalized
SELECT
 lower(gram),
 year,
 occurrences
FROM
 english_1grams
WHERE
 year >= 1890 AND
 gram REGEXP "^[A-Za-z+'-]+$";
```

There are a couple of things I'd like to point out here. First, we are filtering out the data before 1890 because there is less of it therefore it is probably noisier.

Second, I'm lower casing the n-gram using the built-in lower() UDF. Finally, I'm using a regular expression to match

Word ratio by decade

Next we want to find the ratio of each word per decade. More books are printed over time, so every word has a tendency to have more occurrences in later decades. We only care about the relative usage of that word over time, so we want to ignore the change in size of corpus. This can be done by finding the ratio of occurrences of this word over the total number of occurrences of all words.

Let's first create a table to store this data. We'll be swapping the year field with a decade field and the occurrence field with a ratio field (now of type double).

```
CREATE TABLE by_decade (
 gram string,
 decade int,
 ratio double
);
```

The query we construct has to first calculate the total number of word occurrences by decade. Then we can join this data with the normalized table in order to calculate the usage ratio. Here is the query.

```
INSERT OVERWRITE TABLE by_decade
SELECT
 a.gram,
 b.decade,
 sum(a.occurrences) / b.total
FROM
 normalized a
JOIN (
 SELECT
  substr(year, 0, 3) as decade,
  sum(occurrences) as total
 FROM
  normalized
 GROUP BY
  substr(year, 0, 3)
) b
ON
 substr(a.year, 0, 3) = b.decade
GROUP BY
 a.gram,
 b.decade,
 b.total;
```

Calculating changes per decade

Now that we have a normalized dataset by decade we can get down to calculating changes by decade. This can be achieved by joining the dataset on itself. We'll want to join rows where the n-grams are equal and the decade is off by one. This lets us compare ratios for a given n-gram from one decade to the next.

```
SELECT
 a.gram as gram,
 a.decade as decade,
 a.ratio as ratio,
 a.ratio / b.ratio as increase
FROM
 by_decade a
JOIN
 by_decade b
ON
 a.gram = b.gram and
 a.decade - 1 = b.decade
WHERE
 a.ratio > 0.000001 and
 a.decade >= 190
DISTRIBUTE BY
 decade
SORT BY
 decade ASC,
 increase DESC;
```

There are a couple things to point out here. First, we filter out any words with a ratio less than or equal to 0.001% of the corpus for the decade we are observing. This blocks out most of the noisy words that only appear a few dozen times per decade, but have massive proportional swings per decade. This ratio was chosen on a whim, but seems to work well.

We are using b.ratio as a denominator. However, we don't have to worry about dividing by zero because the join we are using ensures that b.ratio will always exist. If the word was never used in the previous decade then the join excludes that null row, preventing a divide by zero error. However, this means we will not see any words that are completely new, they had to be used at least once in the previous decade.

Also notice the DISTRIBUTE BY clause. That tells Hive which column should be used as a key to the reducer. Since we only care about the ordering of words within a decade, we don't need to have a total ordering across all decades. This lets Hive run more than one reducer and significantly increases the parallelism. It must still do a total ordering within a decade, but that is much less data.

Note: This query as currently written will send results to standard out. This query returns lots of data and you probably don't want it all dumped into your console. You will likely want to add a LIMIT clause to the end of the query to see just a subset of the data, or send the results into a new table which you can analyze at your leisure.

Results

Here are the top 30 results per decade.

1900
radium, ionization, automobiles, petrol, archivo, automobile, electrons, mukden, anopheles, marconi, botha, ladysmith, lhasa, boxers, suprema, aboord, rotor, turkes, wireless, conveyor, manchurian, erythrocytes, shoare, thirtie, kop, tuskegee, thorium, audiencia, bvo, arteriosclerosis

1910
cowperwood, britling, boches, montessori, venizelos, bolsheviki, salvarsan, photoplay, pacifists, joffre, petrograd, pacifist, bolshevism, airmen, kerensky, foch, boche, serbia, serbian, hindenburg, madero, serbians, bombing, ameen, anaphylaxis, aviators, syndicalism, aviator, biplane, taxi

1920
bacteriophage, fascist, mussolini, fascism, sablin, latvia, insulin, peyrol, volstead, czechoslovakia, iraq, vitamin, kenya, curricular, swaraj, reparations, broadcasting, slovakia, vitamins, gandhi, automotive, kemal, zoning, jazz, isotopes, isoelectric, airscrew, shivaji, czechoslovak, stabilization

1930
dollfuss, goebbels, manchukuo, hitler, sudeten, hitler's, rearmament, nazis, wpa, nazi, nra, manchoukuo totalitarian, pwa, tva, stalin's, peiping, homeroom, kulaks, stalin, devaluation, bta, carotene, broadcasts, corporative, comintern, ergosterol, reichswehr, ussr, businessmen

1940
waveguide, luftwaffe, plutonium, streptomycin, darlan, gaulle, beachhead, lanny, jeeps, penicillin, alamein, radar, bandwidth, psia, thiamine, quisling, sulfathiazole, wpb, airborne, jeep, aftr, bdg, tobruk, pakistan, sulfonamides, evacuees, guadalcanal, airfields, unesco, rommel

1950
qumran, transistors, chlorpromazine, transistor, automation, terramycin, chloramphenicol, khrushchev, reserpine, pradesh, nasser, vietnamese, shri, uttar, madhya, vietnam, adenauer, aureomycin, nato, annexure, dna, edc, rna, biophys, pyarelal, cortisone, semiconductors, rajasthan, minh

1960
tshombe, bhupesh, vietcong, lumumba, ribosomal, lasers, ribosomes, ieee, aerospace, malawi, thant, fortran, zambia, medicare, lysosomes, nlf, laser, tanzania, efta, oecd, astronaut, teilhard, goldwater, programed, uar, software, autoimmune, spacecraft, eec, nasa

1970
biofeedback, sexist, sexism, multinationals, namibia, bangladesh, microprocessor, watergate, chicano, lifestyle, cytosol, medicaid, trh, chicanos, plasmid, jovanovich, ldcs, apg, pediatr, cyclase, isbn, immunotherapy, prostaglandin, opec, prostaglandins, gentamicin, bangla, radioimmunoassay, epa, ophthalmol

1980
htlv, dbase, interleukin, spreadsheet, vlsi, videotex, calmodulin, sandinistas, contras, isdn, gorbachev's, sandinista, gorbachev, workstation, workstations, fsln, captopril, hybridoma, ifn, robotics, kda, fibronectin, khomeini, sql, robotic, oncogenes, rajiv, xiaoping, unix, microsoft

1990
netscape, cyberspace, html, endothelin, toolbar, biodiversity, mpeg, tqm, harpercollins, applet, reengineering, nafta, http, c++, newsgroups, gallopade, belarus, internet, apec, url, yeltsin, adhd, apoptosis, integrin, usenet, hypermedia, globalisation, netware, africanamerican, myanmar

2000
bibliobazaar, itunes, cengage, qaeda, wsdl, aspx, xslt, actionscript, xpath, sharepoint, blogs, easyread, ipod, xhtml, blog, rfid, google, writeline, proteomics, bluetooth, voip, microarray, mysql, microarrays, putin, dreamweaver, dvds, ejb, xml, osama

Next steps

Now that we've seen how to calculate a rough estimate of popular words, what can we do to enhance our results? Here are some possible avenues for additional research.

- Extend it to search bigrams (2-grams) for popular phrases.
- Switch to 5 year increments to try and find words whose growth spanned decades.
- The current algorithm will not find entirely new words. The word has to have appeared at least once in the prior decade in order to show up in results, which might miss some important words. Try switching the query to use a LEFT OUTER JOIN and surface those words which are completely new.
- Adjust the ranking function to take into account absolute increase in word popularity and not just relative increase.
- Improve the noise filter algorithm. Instead of throwing away any grams that have a fixed ratio, can we just throw out the bottom n percent? How do we determine n?
- Add some basic stemming to improve the results. Now we get nazi/nazis and fascism/fascist. Can we use

stemming to find the roots of these words and return just a summary of topics?

Free to join. Only pay for what you use. Sign Up

Learn

Products & Services
Case Studies
Economics Center
Architecture Center
Security Center
Whitepapers
Videos & Webinars
Industry Solutions
Use Case Solutions
User Groups
AWS Marketplace
Partners

Developer Resources

AMI Catalog
Sample Code & Libraries
SDKs & Tools
Documentation
Articles & Tutorials
Management Console
Flexible Payments Service

Developer Centers

Java
JavaScript
Mobile
PHP
Python
Ruby
Windows & .NET

Manage Your Account

Management Console
Account Activity
Usage Reports
Personal Information
Payment Method
AWS Identity & Access Management
Security Credentials
Request Service Limit Increases

Support

AWS Support
Service Health Dashboard
Discussion Forums
FAQs
Contact Support

About AWS

What is Cloud Computing?
Events
Careers at AWS
Contact Us
Announcements (What's New?)
AWS Blog
Press Releases
Media Coverage
Legal