A1 Sample Solutions and Grading Guide

Marti Hearst

Below are my guidelines for assigning points as well as my solutions, but I suspect that students will be very creative and come up with their own great ideas for how to solve these problems.

Q1: The purpose of this question was to help students learn how Pig works. They are to demonstrate a clear understanding of the underlying commands. They should illustrate the concepts by describing the schema or the data in the schema when answering the questions, but if they come up with some other creative way to answer, then that is fine too. I expect most people will get all 10 points on this one.

Q2: See my solutions below. People may not get the exact same answers that I did if they processed the data differently than I did. For instance, they might have written their own code to find blank queries and not used the code to remove URLs.

Since there are 8 questions and 10 points, here is how to grade: If the student finds a correct way to do all the questions, then they get 10 points. If they don't get some of the questions, subtract one point for each question not gotten. Take off half points if they only partially get a question right.

Q3: This was a bonus question. If they write a UDF to do the job, give them 3 extra points. Total on the whole assignment is 35 points, but make a special note if they do this question and write some congratulatory language! Also, please find a few good samples to share with the class.

Q4: See my samples below. Here is where you have leeway to get more points for better answer. I did a very simple job at this. I would give my answer 10 points out of the 15 since I have a lot of false positives. Some people did really great jobs going the extra mile and we should give them extra points for that extra work.

```
-- QUESTION 2
-- These commands are from the original tutorial
REGISTER ./tutorial.jar;
raw = LOAD 'excite-small.log' USING PigStorage('\t') AS (user, time, query);
clean1 = FILTER raw BY org.apache.pig.tutorial.NonURLDetector(query);
clean2 = FOREACH clean1 GENERATE user, time, org.apache.pig.tutorial.ToLower(query) as query;
clean3 = FOREACH clean2 GENERATE user, time, TOKENIZE(query) as query;


-- Compute the number of words in each query; store it with the query; to be used later
clean4 = FOREACH clean3 GENERATE user, time, query, SIZE(query) as query_len;

-- QUESTION 2A: Count the total number of records
```

```
num_records = FOREACH (GROUP clean4 ALL) GENERATE COUNT(clean4) as ct;
dump num_records;
```

(3882)

```
-- QUESTION 2B: Get ready to get the max query length
ordered_by_qlength = ORDER clean4 BY query_len desc;

-- QUESTION 2B:  Get the max query length
max_qlen = FOREACH (GROUP clean4 ALL) GENERATE MAX(clean4.query_len);
dump max_qlen;
```

(14)

```
-- QUESTION 2B: Check this result (optional)
long_queries = FILTER clean4 by query_len > 13;
dump long_queries;
```

(19887D73626B5D55,970916211949,{(the),(comedy),(of),(errors),(i),(to),(the),(world),(am),(like),(a),(drop),(of),(water)},14)

```
-- QUESTION 2C: Get the average query length
avg_qlen = FOREACH (GROUP clean4 ALL) GENERATE AVG(clean4.query_len);
dump avg_qlen;
```

(2.445388974755281)

```
-- QUESTION 2D: Prepare to get the number of unique users
users = FOREACH clean4 GENERATE user;

-- QUESTION 2D: Prepare to get the number of unique users
num_users = FOREACH (GROUP users ALL) GENERATE COUNT(users);

-- QUESTION 2D: Remove the repeats
distinct_users = DISTINCT users;

-- QUESTION 2D: Count them up
num_distinct_users = FOREACH (GROUP distinct_users ALL) GENERATE COUNT(distinct_users) as ct;
dump num_distinct_users;
```

(842)

```
-- QUESTION 2E: Get the average number of query records per user

avg_recs_per_user = FOREACH num_distinct_users GENERATE (double) num_records.ct / (double) num_distinct_users.ct;
dump avg_recs_per_user;
```

(4.610451306413301)

```
-- QUESTIONS 2F-H: preparation: Get the unique queries
queries = FOREACH clean4 GENERATE query, query_len;
distinct_queries = DISTINCT queries;
```

```
total_distinct_queries = FOREACH (GROUP distinct_queries ALL) GENERATE
COUNT(distinct_queries) as ct;

-- Get the queries that contain a plus sign
-- Need to use clean2 version since that is before tokenization, to operate on a string rather than a
tuple
-- It's easiest to use MATCHES

have_plus = FILTER clean2 BY query MATCHES '.*\\+.*';
total_have_plus = FOREACH (GROUP have_plus ALL) GENERATE COUNT(have_plus);
dump total_have_plus;

 (142)

-- Alternatively, if you used REGEX_EXTRACT, it generates empty results if no matches, so then you
have to filter

matchresults = FOREACH clean2 GENERATE REGEX_EXTRACT_ALL(query, '(.*)(\\+)(.*)') as match;
have_plus = FILTER matchresults BY SIZE(match) > 1;
total_have_plus = FOREACH (GROUP have_plus ALL) GENERATE COUNT(have_plus);

-- Now find all queries with Boolean operators;
-- This does allow for double-counting so may want to run DISTINCT over i.

have_bool = FILTER clean2 BY query MATCHES '.*(\\+| and | or | not ).*';
distinct_have_bool = DISTINCT have_bool;
total_have_bool = FOREACH (GROUP distinct_have_bool ALL) GENERATE COUNT(distinct_have_bool)
as ct;
dump total_have_bool;

 (280)

-- Now compute the percent

percent_bool = FOREACH total_have_bool GENERATE (double) total_have_bool.ct / (double)
total_distinct_queries.ct ;

  (0.13847675568743817)

-- QUESTION 2G: Get the length of the queries in order
ordered_by_qlength = ORDER distinct_queries BY query_len desc;

-- QUESTION 2G: The top 10 longest queries
top_10_qlength = LIMIT ordered_by_qlength 10;
dump top_10_qlength;

  ({(the),(comedy),(of),(errors),(i),(to),(the),(world),(am),(like),(a),(drop),(of),(water)},14)

({(taper),(optic),(tapers),(fiber),(fibre),(waveguide),(optics),(waveguides),(cladding),(optical),(phot
onics)},11)
  ({(iron),(and),(steel),(+oil),(and),(gas),(pipelines),(+valves),(+flanges),(+russia),(+ukraine)},11)
  ({(older),(and),(mature),(and),(women),(and),(pics),(and),(free),(and),(amature)},11)

({(alanta),(georgia/contractors),(independent),(excavation),(contractor),(excavating),(subcontracto
rs),(construction),(earthwork),(bidding)},10)
```

```
  ({(information),(on),(bed),(and),(breakfast),(property),(for),(sale),(in),(canada)},10)
  ({(essays-),(why),(can't),(we),(just),(be),(friends),(the),(female),(perspective)},10)
  ({(i),(to),(the),(world),(am),(like),(a),(drop),(of),(water)},10)
  ({(the),(best),(of),(woman),(&),(ass),(&),(400),(free),(pics)},10)
  ({(salmon),(fishing),(in),(port),(ontario),(ny),(sportfishing),(fly),(river)},9)


-- QUESTION 2H: The 10 most frequently occurring queries
-- Here we are allowing the same users to repeat as often as they like

by_query = GROUP clean2 BY query;

query_counts = FOREACH by_query GENERATE group as query, COUNT(clean2) as count;

ordered_query_counts = ORDER query_counts BY count desc;

top_10_frequent_queries = LIMIT ordered_query_counts 10;
dump top_10_frequent_queries;

  (maytag,41)
  (vanderheiden,27)
  (change bowel habits,24)
  (en vogue,23)
  (running shoes,22)
  (pregnant,20)
  (ebony divas black,19)
  (the byker wall,16)
  (yahoo chat,16)
  (jarrow,16)

-- QUESTION 4: Learn how to make use of another data file.
-- The main goal of this question is to have you do a simple join.

-- Start with commands from the original tutorial
REGISTER ./tutorial.jar;
raw = LOAD 'excite-small.log' USING PigStorage('\t') AS (user, time, query);
clean1 = FILTER raw BY org.apache.pig.tutorial.NonURLDetector(query);
clean2 = FOREACH clean1 GENERATE user, time, org.apache.pig.tutorial.ToLower(query) as
query;
clean3 = FOREACH clean2 GENERATE user, time, TOKENIZE(query) as query;

-- First download the geonames file.  It is tab separated.
-- I focus only on the second column in the geonames file.
-- Note the use of chararray to convert the type
-- Note also I ignore everything after the second column.

geonames_raw = LOAD 'geonames.txt' USING PigStorage('\t') AS (x:chararray,
geoname:chararray);

-- Convert the text to lowercase so it will match the queries.
```

```
geonames_all = FOREACH geonames_raw GENERATE LOWER(geoname) AS geoname;

-- There are a lot of false positives.  I decided to remove really short place names like "at"

geonames = FILTER geonames_all BY SIZE(geoname) > 4;

-- Here we match the entire query
clean5 = FOREACH clean2 generate query, SIZE(query) as querylen;
clean6 = DISTINCT clean5;
queries_with_geonames = JOIN clean5 by query, geonames by geoname;
dist_queries_with_geonames = DISTINCT queries_with_geonames;

ordered_queries_with_geonames = ORDER dist_queries_with_geonames BY querylen DESC;

num_matches = FOREACH (GROUP dis_ordered_queries_with_geonames ALL) GENERATE
COUNT(dis_ordered_queries_with_geonames) as ct;
dump num_matches;

 (70)

-- Lets look at a few.  Many are unfortunately not really placenames.
top10 = LIMIT ordered_queries_with_geonames 10;
dump top10;

 (fredericksburg,14,fredericksburg)
 (south shields,13,south shields)
 (port douglas,12,port douglas)
 (kansas city,11,kansas city)
 (afghanistan,11,afghanistan)
 (hot springs,11,hot springs)
 (tuxedo park,11,tuxedo park)
 (portishead,10,portishead)
 (maastricht,10,maastricht)
 (louisville,10,louisville)

-- Here we break queries into single words and then match
-- The FLATTEN command here converts the QUERY tuple into a record for each query so
the
-- join will work.  I keep the user id but get rid of the time to reduce repeat queries.
-- Here I am only matching single-word terms.  I could also run this on the output of the
ngram
-- parser from the tutorial to match two-word place name as well.

flat_clean3 = FOREACH clean3 GENERATE user, query, FLATTEN(query) as query_term;

-- Now we get to do our join!  This finds queries that contain the geoname.

queries_with_geonames = JOIN flat_clean3 by query_term, geonames by geoname;

distinct_queries_with_geonames = DISTINCT queries_with_geonames;
```

```
-- Lets look at a few.  Many are unfortunately not really placenames.
top10 = LIMIT distinct_queries_with_geonames 10;

-- We also want to match queries that contain zip codes, using a regular expression.
-- Unfortunately, this only matches a tv show in this collection!

have_zip = FILTER clean2 BY query MATCHES '.*[0-9]{5}.*';
dump have_zip;
```