Pig assignment
(revised 9/10/2012; total points now 35)

Readings to prepare for this assignment:

- Programming Pig, Chapters 1-5
  http://ofps.oreilly.com/titles/9781449302641/
- Pig Documentation, Tutorial
  http://pig.apache.org/docs/r0.9.2/start.html#tutorial

Reference Materials

- Pig Documentation, Built-In Functions
  http://pig.apache.org/docs/r0.9.2/func.html
- Pig Documentation, Pig Latin Basics
  http://pig.apache.org/docs/r0.9.2/basic.html
- Java Regular Expression Syntax
  http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html

Assignment Goal

The goal of this assignment is to get you familiar with Pig and the Pig Latin language, both with how to program it and (hopefully) with how to run it on a Hadoop cluster to see the speed advantages. You may then want to use this functionality in your final project or in future data analysis work.

Background

As we learned in the readings, currently people who need to wrestle with big data sets often make use of software systems like Hadoop (an open source implementation of the MapReduce system) because these systems handle the complexity of efficient and reliable computation across many computers, disks, and local networks transparently for the programmer. However, Hadoop has a rather restrictive language model, and therefore, several programming languages have arisen to make it easier to program distributed computational systems like Hadoop.

For this assignment, we are learning to use one of those languages, called Pig Latin, since it is heavily used at Twitter. (This language is brand new to me, by the way. I'm finding it quite fun to learn!) Pig Latin is an interesting blend of scripting and SQL-like functions (joins, groups, etc), and provides an interesting alternative to a SQL-like language for data processing. It also provides an easy way for users to integrate their own functions into the language.

Tips: Viewing Status

Pig doesn't make it easy for typical debugging print statements, but provides some alternatives. Be sure to make use of the following commands for viewing the relations you've built:

- DESCRIBE
- ILLUSTRATE

You can use the DUMP command to see the results so far dumped to the terminal (STDOUT), but that is often a bad idea if there is a lot of data. One trick to make this more time efficient is to use the LIMIT command. For example:

```
just_10 = LIMIT data 10;
DUMP just_10;
```

Another useful built in function is SAMPLE, which lets you create a random subset of the data for quick iteration. The data, while not gigantic, is large enough that developing a script on the full data set would take a long time. This "sample, iterate, scale" approach is what they use at Twitter to build production Pig scripts.

First, we want a really small subset (~20 rows) to iterate quickly on.

```
just_1pct = SAMPLE data 0.01;
just_20 = LIMIT just_1pct 20;
store just_20 into 'tiny_sample';
store just_1pct into 'small_sample';
```

You can now use the follow load statements:

tiny_sample = load 'tiny_sample' as (entity:chararray, text:chararray);
small_sample = load 'small_sample' as (entity:chararray, text:chararray);

Step 1: Learn the Tutorial

(10 points) Below I've annotated the steps of the Pig documentation tutorial as a way to help you learn Pig Latin. By stepping through it with me and answering these questions I'm hoping this will help you learn the details and prepare you to write your own code. Be sure to answer the questions I've highlighted below!

Let's look at the log file tutorial.
http://pig.apache.org/docs/r0.9.2/start.html#tutorial

(In my installation, the release notes said I had to run "ant" first to build the tutorial correctly; it creates a new directory. You might need to copy the excite-small.log file over from the test directory; I did.)

In each case I retain the original comment from the tutorial and then add my own comments.

Register the tutorial JAR file so that the included UDFs (user-defined functions, in this case defined by the writer of the tutorial) can be called in the script. You can see the code for the user-defined functions in the src directory within the tutorial directory, for instance in pig-0.10.0/tutorial/src/org/apache/pig/tutorial/NonURLDetector.java

```
REGISTER ./tutorial.jar;
```

Use the PigStorage function to load the excite log file (excite.log or excite-small.log) into the "raw" bag as an array of records with the fields user, time, and query. The '\t' tells PigStorage to divide up each line, using the tab character as a delimiter. There are only 3 fields in the file, so the first field becomes the user ID, the second the time field, and the third, the query field. (Presumably the queries do not contain tabs

embedded within them.) By default, the fields are referred to as fields numbered $0 $1, and $2. The AS operator tells Pig Latin to assign the names user, time, and query to those fields.

```
raw = LOAD 'excite.log' USING PigStorage('\t') AS (user, time, query);
```

Call the NonURLDetector UDF to remove records if the query field is empty or a URL. The FILTER command retains the records that match the condition in the BY field, or more precisely, for which the BY condition returns TRUE.

```
clean1 = FILTER raw BY org.apache.pig.tutorial.NonURLDetector(query);
```

Call the ToLower UDF to change the query field to lowercase. Note that in this case, the statement retains the first two fields from the previous tuples (user, time) and restates them here, in the same order. The programmer could have opted to change the order, dropped one of them, etc. For the third field, the query is transformed by lowercasing it.

```
clean2 = FOREACH clean1 GENERATE user, time, org.apache.pig.tutorial.ToLower(query) as query;
```

Because the log file only contains queries for a single day, we are only interested in the hour. The excite query log timestamp format is YYMMDDHHMMSS. Call the ExtractHour UDF to extract the hour (HH) from the time field. Alternatively, you could use a regular expression to extract out the hour from this field. We'll be using regular expressions in the assignment below.

```
houred = FOREACH clean2 GENERATE user, org.apache.pig.tutorial.ExtractHour(time) as hour, query;
```

Call the NGramGenerator UDF to compose the n-grams of the query. An ngram usually refers to a sequence of n adjacent words, but it can also refer to a sequence of n adjacent characters (so a bigram is two adjacent words, a trigram is three adjacent words). Here it seems to be used without specifying a value for n, which is nonstandard. A look at the

code for the UDF shows that it is hard-coded with N=2, and the code generates ngrams of all lengths from 1 up to and including N.

Why are we doing this? The idea is to find the most common single words and pairs of words used across all user queries. Below we will count them up. Here we list the unigrams and bigrams for a given line from the log. We also show the user and the hour.

```
ngramed1 = FOREACH houred GENERATE user, hour,
flatten(org.apache.pig.tutorial.NGramGenerator(query)) as ngram;
```

Use the DISTINCT operator to get the unique n-grams for all records.
Question 1.a: Why do we do this? Under what circumstances would there be duplicates?

```
ngramed2 = DISTINCT ngramed1;
```

Use the GROUP operator to group records by n-gram and hour. The GROUP operator takes the arguments in the BY operand and uses those ask the keys to group by. It places those in position $0 of the resulting record. The rest of the original record goes in position $1 of the resulting record. For more information, see http://pig.apache.org/docs/r0.9.2/basic.html#GROUP
Question1.b In English, what is this command doing?

```
hour_frequency1 = GROUP ngramed2 BY (ngram, hour);
```

Use the COUNT function to get the count (occurrences) of each n-gram.
Question 1.c Why is the FLATTEN command used here? What are we really counting with COUNT($1) mean? (Hint: see the GROUP documentation linked to above.)

```
hour_frequency2 = FOREACH hour_frequency1 GENERATE
flatten($0), COUNT($1) as count;
```

Use the GROUP operator to group records by n-gram only. Each group now corresponds to a distinct n-gram and has the count for each hour. The :: is a disambiguation operator which is used to identify fields after JOIN, GROUP, FLATTEN, etc. The first field the results after a GROUP is

(confusingly) called "group", and that is what is being referred to below.

Question 1.d In English, what is this command doing?

```
uniq_frequency1 = GROUP hour_frequency2 BY group::ngram;
```

For each group, identify the hour in which this n-gram is used with a particularly high frequency. Call the ScoreGenerator UDF to calculate a "popularity" score for the n-gram. This returns four values: hour, score, count, and mean.

```
uniq_frequency2 = FOREACH uniq_frequency1 GENERATE flatten($0), flatten(org.apache.pig.tutorial.ScoreGenerator($1));
```

Use the FOREACH-GENERATE operator to assign names to the fields.

```
uniq_frequency3 = FOREACH uniq_frequency2 GENERATE $1 as hour, $0 as ngram, $2 as score, $3 as count, $4 as mean;
```

Use the FILTER operator to move all records with a score less than or equal to 2.0. (Actually retain only those records that are greater than 2.0.)

```
filtered_uniq_frequency = FILTER uniq_frequency3 BY score > 2.0;
```

Use the ORDER operator to sort the remaining records by hour and score.

```
ordered_uniq_frequency = ORDER filtered_uniq_frequency BY hour, score;
```

Use the PigStorage function to store the results. The output file contains a list of n-grams with the following fields: hour, ngram, score, count, mean. If you can't write to /tmp, you can write instead to a local directory 'tutorial-results'.

```
STORE ordered_uniq_frequency INTO '/tmp/tutorial-results' USING PigStorage();
```

Step 2: Compute Overall Query Log Statistics

(**10 points**). Now its time for you to write some code of your own. For reporting results, please use the excite-small.log file although you're free to run this on the larger file. Below, I use the term query record to refer to a (user, time, query) triplet.

Write a script that outputs:

- For all query records with **non-empty** queries:
  a) The total number of query records
  b) The maximum query length in words
  c) The average query length in words
  d) The total number of unique users
  e) The average number of query records per user
  f) What percent of query records contain queries with Boolean operators (AND, OR, NOT, or +)
  g) The 10 longest distinct queries in words
  h) The 10 most frequently occurring queries

Note by the way that TOKENIZE is the a built in function that separates strings into words, and returns a Bag of words. LOWER is also a useful built in function.

Step 3: Compute User Session Information (Optional Bonus Question)

~~**(15 points)**~~ The following text is from my book, *Search User Interfaces*.

> "Information seeking is often a complex process consisting of many steps. For this reason, many studies of query logs attempt to infer searchers' intent by observing their behavior across a sequence of steps or activities.

> This sequence is usually referred to as a *search session*, which can be defined as a sequence of requests made by a single user for a single navigation purpose (Huang et al., 2004).
>
> There are several approaches for automatically distinguishing the boundaries of search sessions. The simplest method is to set a time threshold; when a user has been inactive for some amount of time, a cutoff is imposed, and all actions that happened before the cutoff are grouped into a session. He and Goker, 2000 found a range of 10 to 15 minutes of inactivity to be optimal for creating session boundaries for web logs. Silverstein et al., 1999 used five minutes, Anick, 2003 used 60 minutes, and Catledge and Pitkow, 1995 recommended 25.5 minutes.
>
> Chen et al., 2002 recognize that timing information varies with both user and task, and so proposed an adaptive timeout method. They defined the "age" of an action to be the interval between an action and the current time. A session boundary was assigned when the age of an action was two times older than the average age of actions in the current session. Thus the time that is allowed to elapse before a session boundary is declared varies with the degree of activity the user is engaged in at different points in time."

For this part of the assignment, I'd like you to write Pig code to recognize session boundaries in the Excite query logs. You must describe the algorithm you are using, and motivate it with the literature referred to above, or with some other good motivation if you don't use those sources.

Write a script that computes, for query records with **non-empty** queries

a) For each user, how many query sessions that user has (but don't print this out)
b) The maximum query session length
c) The average query session length
d) The standard deviation of query session lengths.

~~BONUS:~~

e) Instead of fixed session lengths, write a script that computes dynamic session lengths as described by Chen et al above.

Step 4: Combine with Another Data Set

(15 points) Now let's do some analysis of the content of the queries. Many queries contain place names and geographic terms. For this part of the assignment, your job is to make a stab at figuring out how many queries contain one or more references to place names. I'd like you to do this at least two different ways:

a) Find queries with references to U.S. zip codes
b) Find queries with references to place names

BONUS

c) Find other indicators of geographic locations.
d) Further improve on (b) by doing something clever about ambiguous place names.

To find references to zip codes, you should use regular expression matching. You can use REGEX_EXTRACT_ALL, or the MATCHES command paired with FILTER is great for finding regular expression matches within strings.

```
hasrun = FILTER clean2 BY query MATCHES '.*run.*';
```

Note that if you use a regular expression containing a \ you have to escape it with a second one, for example, '\\d'

To find place names, the first thing to do is get a list of place names! I recommend the World Gazeteer, which I've downloaded and tested out:
http://www.world-gazetteer.com/wg.php?x=1129163518&men=stdl&lng=en&gln=xx&dat=32&srt=npan&col=aohdq

You can use PigStorage() to parse it as you read it in. Be sure to watch the data types! You may need to transform it (you can use the UDFs from the tutorial, or Pig built-ins like TOKENIZE and LOWER).

I'm not looking for perfection here. Some place names are also regular words, and so will erroneously match regular words.

Homework Format; Turning in the Assignment

We will be putting up a link for you to turn in your code and data at. We will expect you to turn in your code and well-formatted output that refers to the question numbers (e.g., 2(a), 3(b)). We also expect a well-written write up that explains your algorithms where appropriate.

You are to do your own work, but you may discuss this assignment with other students and the TAs and instructors. We discourage you from looking for solutions on the web other than from the Pig documentation pages and other assigned reading. If you do find code from any outside source you are required to document where you found that code from.

Bonus points can put you up to but not beyond 100%.

Epilogue

Given the size of the data set (not gigantic), you may wonder what the benefit of learning Pig is, compared just to writing a quick script in a language such as Python or R. The key point is this: the script you wrote is potentially scalable to Petabytes of data. All you need is a cluster, and the same script will work. The framework is less useful for smaller datasets, but being able to scale up your analysis to almost arbitrary sizes is something that before Pig was essentially impossible.

Acknowledgements

Parts of this assignment written by @jco, and he had help from @J_, @THISWILLWORK, @squarecog, @billgraham