CS-5340/6340, Programming Assignment #3
Due: Thursday, Nov. 3, 2016 by 11:00pm

For this assignment, you should write a program that implements a simplified version of the Basilisk bootstrapping algorithm for semantic lexicon induction. We will call this program *Basilisk-Lite*. As input, your program will need two files: (1) a file of seed words for a semantic category, and (2) a file of context data from a text corpus. Your program should read the names of the two input files from the command line. The command-line arguments should be in the following order:

```
basilisk <seeds_file> <contexts_file>
```

For example, we should be able to run your program like so:

```
basilisk seeds.txt contexts.txt
```

1. The Seeds File

This file will contain a list of seed words for the semantic category to be learned. Each word will be on a separate line, and each line will contain just one word. For example, a seeds file for ANIMALS might look like this:

cat
dog
snake
frog
sparrow

Your program should be case insensitive, so "cat", "Cat", and "CAT" should all be treated as the same token.

2. The Contexts File

This file will contain pattern contexts that have been generated from a text corpus. Each line will consist of 3 parts:

1. **noun phrase** : a sequence of words that comprise a noun phrase
2. ***** : an asterisk will separate the noun phrase from the pattern
3. **pattern** : a single token that represents a specific linguistic context

A sample file might look like this:

> CASTELLAR * <subj>__PassVp:POS__KIDNAPPED
> ELN GUERRILLAS * PassVp_Prep_<NP>:POS__KIDNAPPED_BY
> HE * <subj>_ActVp:POS__TRAVELING
> A BOAT * ActVp_Prep_<NP>:POS__TRAVELING_IN
> THE CAUCA RIVER * ActVp_Prep_<NP>:POS__TRAVELING_DOWN
> THE TENCHE AREA * ActVp_Prep_<NP>:POS__TRAVELING_TO

It's not essential that you understand exactly what each pattern context represents, but fyi, each pattern token consists of three parts: (a) syntactic template, (b) POS [positive] or NEG [negative] indicator, (c) specific words that fill in the syntactic template. For example, the pattern <subj>__PassVp:POS__KIDNAPPED represents a passive voice verb phrase (PassVP) with no negation (POS) that matched the word KIDNAPPED in a sentence. The noun phrase (CASTELLAR) was found in the SUBJECT (<subj>) position of the sentence. We will say that the noun phrase "co-occurs" with the pattern. The pattern contexts shown above were generated from the sentence:

> CASTELLAR WAS KIDNAPPED BY ELN GUERRILLAS WHILE HE WAS TRAVELING IN A BOAT DOWN THE CAUCA RIVER TO THE TENCHE AREA.

Your program should be case insensitive, so "castellar", "Castellar", and "CASTELLAR" should all be treated as the same token.

NOTE: the context files that we give you have been *automatically* generated by a parser and pattern generation tool. These tools are not perfect, so you may encounter output that doesn't seem correct (e.g., a so-called noun phrase that isn't really a noun phrase). This is the reality of natural language processing in the wild!

OUTPUT FORMATTING

The output produced by your program should consist of the following items:

1. A list of the seed words read from the seeds file.
2. The number of unique patterns found in the context file.
3. For each iteration:
   a) Print the list of the patterns chosen for the Pattern Pool, with the RlogF score for each pattern shown in parentheses.
   b) Print the new words selected for the semantic lexicon, with the AvgLog score for each word shown in parentheses.

IMPORTANT: for the sake of consistency in your output files, please do the following:

- For both the RlogF and AvgLog scores, print the numbers with exactly 3 digits after the decimal point.
- When multiple patterns have the same RlogF score, print them in alphabetical order.
- When multiple words have the same AvgLog score, print them in alphabetical order.
- Please format your output exactly like the example shown on the next page.

EXAMPLE OUTPUT FILE

```
Seed Words: people guerrillas members troops cristiani rebels president terrorists soldiers lea
Unique patterns: 20982

ITERATION 1

PATTERN POOL
1. <subj>_ActVp:POS__BLEW_UP  (1.203)
2. PassVp_Prep_<NP>:POS__ATTACKED_BY  (1.203)
3. Np_Prep_<NP>:POS__PRESENCE_OF  (1.088)
4. <subj>_ActVp:POS__PLACED  (1.080)
5. <subj>_ActVp:POS__BURNED  (1.057)
6. <subj>_ActVp:POS__COMBING  (1.057)
7. <subj>_ActVp:POS__HIDING  (1.057)
8. <subj>_ActVp:POS__REQUEST  (1.057)
9. <subj>_ActVp:POS__TRANSPORTED  (1.057)
10. <subj>_ActVp_Dobj:POS__ATTACKED_TROOPS  (1.057)
11. <subj>_ActVp_Dobj:POS__COMBING_AREA  (1.057)
12. <subj>_ActVp_Dobj:POS__SUSTAINED_CASUALTIES  (1.057)
13. <subj>_AuxVp_Dobj:POS__BE_COMMANDER  (1.057)
14. ActVp_Prep_<NP>:POS__DELIVER_TO  (1.057)
15. ActVp_Prep_<NP>:POS__MURDERED_BY  (1.057)
16. Subj_AuxVp_<dobj>:POS__BE_MEMBERS  (1.057)

NEW WORDS
advisors  (2.322)
contingents  (2.322)
observers  (1.953)
futh  (1.333)
commandoes  (1.000)

ITERATION 2

PATTERN POOL
1. Np_Prep_<NP>:POS__PRESENCE_OF  (1.904)
2. <subj>_ActVp:POS__BLEW_UP  (1.604)
3. <subj>_ActVp:POS__BURNED  (1.409)
4. PassVp_Prep_<NP>:POS__ATTACKED_BY  (1.203)
5. <subj>_ActVp:POS__PLACED  (1.080)
6. <subj>_ActVp:POS__COMBING  (1.057)
7. <subj>_ActVp:POS__HIDING  (1.057)
8. <subj>_ActVp:POS__REQUEST  (1.057)
9. <subj>_ActVp:POS__TRANSPORTED  (1.057)
10. <subj>_ActVp_Dobj:POS__ATTACKED_TROOPS  (1.057)
11. <subj>_ActVp_Dobj:POS__COMBING_AREA  (1.057)
12. <subj>_ActVp_Dobj:POS__SUSTAINED_CASUALTIES  (1.057)
13. <subj>_AuxVp_Dobj:POS__BE_COMMANDER  (1.057)
```

14. ActVp_Prep_<NP>:POS__DELIVER_TO (1.057)
15. ActVp_Prep_<NP>:POS__MURDERED_BY (1.057)
16. Subj_AuxVp_<dobj>:POS__BE_MEMBERS (1.057)

NEW WORDS
cyanide (1.000)
&&4 (0.943)
narcoterrorists (0.821)
demonstrators (0.802)
employees (0.800)

ITERATION 3

PATTERN POOL
1. Np_Prep_<NP>:POS__PRESENCE_OF (2.176)
2. PassVp_Prep_<NP>:POS__ATTACKED_BY (2.005)
3. <subj>_ActVp:POS__BURNED (1.761)
4. <subj>_ActVp:POS__BLEW_UP (1.604)
5. <subj>_ActVp:POS__TRANSPORTED (1.585)
6. <subj>_ActVp:POS__ATTACKED (1.340)
7. <subj>_ActVp:POS__SUSTAINED (1.203)
8. <subj>_ActVp:POS__PLACED (1.080)
9. <subj>_ActVp:POS__COMBING (1.057)
10. <subj>_ActVp:POS__HIDING (1.057)
11. <subj>_ActVp:POS__REQUEST (1.057)
12. <subj>_ActVp_Dobj:POS__ATTACKED_TROOPS (1.057)
13. <subj>_ActVp_Dobj:POS__COMBING_AREA (1.057)
14. <subj>_ActVp_Dobj:POS__SUSTAINED_CASUALTIES (1.057)
15. <subj>_AuxVp_Dobj:POS__BE_COMMANDER (1.057)
16. ActVp_Prep_<NP>:POS__DELIVER_TO (1.057)
17. ActVp_Prep_<NP>:POS__MURDERED_BY (1.057)
18. Np_Prep_<NP>:POS__WORDS_AS (1.057)
19. Subj_AuxVp_<dobj>:POS__BE_MEMBERS (1.057)

NEW WORDS
nicaraguan (2.377)
fighters (1.042)
mercenaries (1.034)
artillery (0.792)
children (0.781)

ITERATION 4

PATTERN POOL
1. Np_Prep_<NP>:POS__PRESENCE_OF (2.176)
2. <subj>_ActVp:POS__BURNED (2.113)
3. <subj>_ActVp:POS__ATTACKED (2.010)
4. PassVp_Prep_<NP>:POS__ATTACKED_BY (2.005)

5. <subj>_ActVp:POS__BLEW_UP (1.604)
6. <subj>_ActVp:POS__TRANSPORTED (1.585)
7. <subj>_ActVp:POS__SUSTAINED (1.203)
8. <subj>_ActVp:POS__PLACED (1.080)
9. <subj>_ActVp:POS__COMBING (1.057)
10. <subj>_ActVp:POS__FALLEN (1.057)
11. <subj>_ActVp:POS__HIDING (1.057)
12. <subj>_ActVp:POS__REQUEST (1.057)
13. <subj>_ActVp_Dobj:POS__ATTACKED_POLICE_STATION (1.057)
14. <subj>_ActVp_Dobj:POS__ATTACKED_TROOPS (1.057)
15. <subj>_ActVp_Dobj:POS__COMBING_AREA (1.057)
16. <subj>_ActVp_Dobj:POS__SUSTAINED_CASUALTIES (1.057)
17. <subj>_AuxVp_Dobj:POS__BE_COMMANDER (1.057)
18. <subj>_PassVp:POS__DEFENDED (1.057)
19. ActVp_Prep_<NP>:POS__DELIVER_TO (1.057)
20. ActVp_Prep_<NP>:POS__MURDERED_BY (1.057)
21. Np_Prep_<NP>:POS__WORDS_AS (1.057)
22. Subj_AuxVp_<dobj>:POS__BE_MEMBERS (1.057)

NEW WORDS
mrta (0.813)
guerillas (0.755)
inmates (0.738)
government_forces (0.732)
columns (0.715)

ITERATION 5

PATTERN POOL
1. <subj>_ActVp:POS__ATTACKED (2.513)
2. PassVp_Prep_<NP>:POS__ATTACKED_BY (2.406)
3. Np_Prep_<NP>:POS__PRESENCE_OF (2.176)
4. <subj>_ActVp:POS__BURNED (2.113)
5. <subj>_ActVp:POS__BLEW_UP (1.604)
6. <subj>_ActVp:POS__SUSTAINED (1.604)
7. <subj>_ActVp:POS__TRANSPORTED (1.585)
8. <subj>_ActVp_Dobj:POS__SUSTAINED_CASUALTIES (1.585)
9. <subj>_ActVp:POS__DETONATED (1.292)
10. ActVp_Prep_<NP>:POS__CLASHED_WITH (1.203)

NEW WORDS
insurgents (0.714)
peasants (0.707)
persons (0.697)
journalists (0.662)
units (0.645)

BASILISK-LITE ALGORITHM

This is the Basilisk-Lite bootstrapping algorithm that you should implement:

Step 1. Initialize the Semantic Lexicon with the seed words: $Lexicon = \{seeds\}$

Step 2. Populate a data structure that stores all of the *unique head nouns* that co-occur with each pattern. For this assignment, assume that the rightmost word in the noun phrase is its head noun. For example, the head noun of "Salt Lake City" would be "City". (This is an imperfect heuristic but will be correct most of the time.)

Example: suppose $pattern_k$ appears in the context data with 5 noun phrases: { *"utah", "beautiful utah", "salt lake city", "mexico city",* and *"boston"*}. Your data structure should contain an entry with the three unique head nouns:

$$pattern_k : \{\ utah,\ city,\ boston\ \}$$

For iterations = 1 to 5 { NOTE: please run for exactly 5 iterations!

Step 3. Assign a score to each pattern using the RlogF formula, where the R value is the probability $P(semclass \mid pattern_k)$ and the F value is the total frequency of the pattern. Specifically:

$$\text{RlogF}(pattern_k) \ = \ \frac{semfreq_k}{totalfreq_k} * log_2(totalfreq_k)$$

where $semfreq_k$ is the number of unique semantic lexicon members that co-occur with $pattern_k$ (i.e., the number of semantic lexicon members that occur in at least one $pattern_k$ context), and $totalfreq_k$ is the total number of unique words that co-occur with $pattern_k$.

Step 4. Sort the patterns by their RlogF score and put the top 10 patterns into a Pattern Pool. If there are ties for the last position, then put all of the patterns with a tied score into the Pattern Pool. But do not put patterns with a score of zero into the Pattern Pool. So the Pattern Pool may have > 10 items in the event of ties, or < 10 items if fewer than 10 patterns have a nonzero score.

Step 5. Collect all of the *unique head nouns* that co-occur with at least one pattern in the Pattern Pool and put them into a Candidate List.

Step 6. Assign a score to each noun in the Candidate List using the AvgLog formula. See the definition and example of the AvgLog function on the next page.

Step 7. Sort the words in the Candidate List by their AvgLog score and add the top 5 *new* words to the Lexicon. (That is, skip over words that are already in the Lexicon when selecting the top 5 new words.) If there are ties for the last position, then add all of the tied words to the Lexicon, so in this case > 5 words may be added.

}

AvgLog Scoring Function

The **AvgLog** scoring function is, essentially, the average number of semantic category members that occur with the patterns found with the candidate word (as a head noun). Except that we average over the log of the frequency values, and add 1 to each frequency value before computing the log (to avoid having to take a log of zero).

To compute $AvgLog(noun_j)$, first collect all of the patterns that co-occur with $noun_j$. (That is, find all of the pattern contexts that have at least one occurrence of $noun_j$ as a head noun.) **Not only the patterns in the Pattern Pool, but all of the patterns!** Let's assume that N patterns are found with the candidate word.

For each of these N patterns, count the number of unique semantic lexicon members that co-occur with the pattern as a head noun (call this $semfreq_i$ for $pattern_i$). Then sum $log_2(semfreq_i + 1)$ over all of the patterns and divide by N.

Example: Suppose the noun "utah" co-occurs with the 8 patterns shown in the table below. The middle column of the table shows all of the head nouns that co-occur with each pattern in the context data. Assume that the Semantic Lexicon currently contains the words: {**city town country neighborhood departments department guatemala colombia medellin bolivia**}. The SemFreq value for each pattern is shown in the third column.

Pattern	Unique Head Nouns	SemFreq
$pattern_1$	**city**, **country**, utah, wyoming	2
$pattern_2$	**medellin**, usa, utah, peru	1
$pattern_3$	utah, france, europe	0
$pattern_4$	**city**, utah, **town**, **country**, **neighborhood**	4
$pattern_5$	utah, ohio, **city**	1
$pattern_6$	**department**,**departments**, **colombia**, utah	3
$pattern_7$	florida, colorado, utah	0
$pattern_8$	colorado, nebraska, minnesota, utah, **town**, **country**	2

The AvgLog score for "utah" would then be:

AvgLog("utah") = ($log_2(2+1)$ + $log_2(1+1)$ + $log_2(0+1)$ + $log_2(4+1)$ + $log_2(1+1)$ + $log_2(3+1)$ + $log_2(0+1)$ + $log_2(2+1)$) / 8

AvgLog("utah") = (1.58 + 1 + 0 + 2.32 + 1 + 2 + 0 + 1.58) / 8

AvgLog("utah") = 1.185

Grading Criteria

Your program will be graded based on new input files! **So please test your program thoroughly to evaluate the generality and correctness of your code!** Even if your program works perfectly on the examples that we give you, that does not guarantee that it will work perfectly on new input.

Please exactly follow the formatting instructions specified in this assignment. We will deduct points if you fail to follow the specifications because it makes our job much more difficult to grade programs that do not conform to the same standards.

IMPORTANT: You may not use ANY external software packages or dictionaries to complete this assignment except for *basic* libraries. For example, libraries for general I/O handling, math functions, and regular expression matching are ok to use. But you may not use libraries or code from any NLP-related software packages, or external code that performs any functions specifically related to the Basilisk algorithm. All submitted code *must be your own.*

If you are unsure about the acceptability of anything, please email us at `teach-cs5340@list.eng.utah.edu`

ELECTRONIC SUBMISSION INSTRUCTIONS

You need to submit 4 things:

1. The source code files for your program. Be sure to include all files that we will need to compile and run your program!

   REMINDER: your program **must** be written in Python or Java, and it **must** compile and run on the Linux-based CADE (lab1 or lab2) machines! We will not grade programs that cannot be run on the Linux-based CADE machines.

2. An executable *shell script* named `basilisk.sh` that contains the exact commands needed to compile and run your basilisk program on the data files provided on the cs5340 web page. We should be able to execute this file on the command line, and it will compile and run your code. For example, if your code is in Python and does not need to be compiled, then your script file might look like this:

   ```
   python basilisk.py seeds.txt contexts.txt
   ```

   If your code is in Java, then your script file might look something like this:

   ```
   javac Basilisk/*.java
   java Basilisk/MainClass seeds.txt contexts.txt
   ```

3. A `README.txt` file that includes the following information:

   - Which CADE machine you tested your program on
     (this info may be useful to us if we have trouble running your program)
   - Any known idiosyncracies, problems, or limitations of your program.

4. Submit one trace file called **basilisk.trace** that shows the output of your program when given the input files on the CS-5340 web page.

You can generate a trace file in (at least) 3 different ways: (1) print your program's output to a file called `basilisk.trace`, (2) print your program's output to standard output and then pipe it to a file (e.g., `basilisk seeds.txt contexts.txt > basilisk.trace`), or (3) print your output to standard output and use the unix *script* command before running your program on the test files. The sequence of commands to use is:

```
script basilisk.trace
basilisk seeds.txt contexts.txt
exit
```

This will save everything that is printed to standard output during the session to a file called `basilisk.trace`.

To turn in your files, the CADE provides a web-based facility for electronic handin, which can be found here:

`https://webhandin.eng.utah.edu/`

Or you can log in to any of the CADE machines and issue the command:

`handin cs5340 basilisk <filename>`

HELPFUL HINT: you can get a listing of the files that you've already turned in via electronic handin by using the 'handin' command without giving it a filename. For example:

`handin cs5340 basilisk`

will list all of the files that you've turned in thus far. If you submit a new file with the same name as a previous file, the new file will overwrite the old one.