CS-6390, Programming Assignment #2
Due: Friday, March 3, 2017 by 11:00pm

For this assignment, you will implement a simplified version of the DIRT algorithm for learning inference rules. Your program should accept the following command-line arguments in this order:

dirt <corpus_file> <test_file> <minfreq>

The command-line arguments are:

corpus_file: the pre-processed text corpus to use for rule discovery.
test_file: a list of phrases for testing.
minfreq: the minimum number of instances required for learned rules.

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

```
dirt corpus.txt test.txt 5
```

---

1. The Corpus File

This file will contain sentences from a text corpus that have been syntactically chunked. Each chunk will be listed on a separate line, with a colon separating the syntactic type from the phrase. For example, a corpus file might look like this:

```
NP : Natural language processing
VP : is
NP : an interesting topic
WORD : >PERIOD
WORD : <EOS
NP : NLP
VP : can be
ADJP : fun
WORD : >PERIOD
WORD : <EOS
```

Each sentence will end with an <EOS (End Of Sentence) token. The corpus files that you will be given have been pre-processed with a tokenizer that changes some of the original words. For example, punctuation marks have been replaced by special tokens (e.g., periods have been replaced by the token >PERIOD) and two ampersands have been appended to the beginning of numbers (e.g., &&3 instead of 3).

The words should be treated as case insensitive, so "cat", "Cat", and "CAT" should all be considered the same token.

2. The Test File

This file will contain phrases that should be used as test cases. Each phrase will be a separate line. For example, a test file might look like this:

> sold
> plans to acquire

The words should be treated as case insensitive, so "cat", "Cat", and "CAT" should all be considered the same token.

THE ALGORITHM

Your program should implement the DIRT algorithm for learning inference rules, as described in the paper *"DIRT - Discovery of Inference Rules from Text"* by Lin and Pantel, which is linked onto our class webpage. For this assignment, the rule representation will be simplified but the DIRT algorithm is essentially the same. Your system should do the following:

- The first step is to create a **Triple Database** of all paths that occur between two noun phrases (NPs) in the corpus. (NOTE: this is different from the paper where paths are constructed between all nouns.) For this assignment, we define a **path** as a sequence of words between two sequential NPs. **SlotX** refers to the NP on the path's left and **SlotY** refers to the NP on the path's right. These noun phrases are called slot fillers.

  You should extract paths (sequences of words) between each pair of two sequential NPs in the same sentence that have at least one (non-NP) word between them. For example, suppose a sentence contains a sequence of 4 NPs, 1 VP, and another NP like this: NP1 NP2 NP3 NP4 VP NP5. In this (bizarre) case, only 1 path should be created for the VP in between NP4 and NP5.

  For each path, record the **head noun** of each NP as the filler for its slot. You can assume that the head noun is simply the rightmost word in each NP.

- **Exclusion List for Illegal Paths:** do not create entries in the Triple Database for the following 1-word paths: *"is", "are", "be", "was", "were", "said", "have", "has", "had", "and", "or", ">COMMA", ">SQUOTE", ">RPAREN", ">LPAREN", ">PERIOD", ">MINUS", ">AMPERSAND"*). These words are very frequent (and therefore require a lot of computation) and are rarely useful inference rules. Note that it is ok for a path to contain these words, so long as the path includes additional words. Only these exact 1-word paths should be excluded.

- For example, consider the following text file:

```
NP : News flash
NP : Professor Riloff
VP : said
NP : information extraction
VP : can be
NP : a fascinating topic
WORD : and
VP : creating
NP : a program
PP : for rule discovery
VP : is
NP : an interesting exercise
WORD : >PERIOD
WORD : <EOS
NP : Each sentence
VP : should be handled separately
WORD : >PERIOD
WORD : <EOS
```

  The Triple Database for this text file should be:

Slot X Filler	**Path**	**Slot Y Filler**
extraction	can be	topic
topic	and creating	program
program	for rule discovery is	exercise

  Note that no path is created for the VP "said" because that 1-word path is in the exclusion list. Also, the second sentence contains only 1 NP, so no paths should be extracted from it.

- **Minfreq Filtering**: after the triple database is created, remove all paths that have < **minfreq** instances in the corpus. DIRT's algorithm is computationally expensive, so we will use this threshold to reduce the number of paths that must be considered. (Beware that computing the similarity of paths can be slow!)

- For each phrase in the test file, your system should find the 5 most similar rules in the Triple Database. Your program should compute the similarity between the test phrase and every path in the triple database (after filtering) using the similarity function described in Section 4.3 of the DIRT paper. Print the 5 most similar paths, along with their similarity scores, as the output (formatting is described in the next section).

- **Special Cases for the MI Function**:
  - the ratio inside the logarithm can lead to undefined values if either the numerator or denominator is zero. If the numerator and/or denominator are zero, then return an MI value of zero.
  - if the logarithm produces a negative value, then return an MI value of zero. (Negative values will cause problems for the subsequent sqrt function.)
  - use log base 2 for the logarithm.

Output Formatting

Your program should print the following output:

1. The number of distinct legal paths found in the corpus file, before and after minfreq filtering.
2. The number of legal path instances found in the corpus file, before and after minfreq filtering.
3. The 5 most similar paths for each phrase in the test file along with their similarity scores, but do not show paths that have a similarity score of zero. In the event of ties for the 5th position, print all paths that have the same score (so in this case you may print more than 5 paths).

A sample (fictitious) output file is shown below. Please format your output to look like this! If a phrase in the test file is not in your triple database, please print the message *"This phrase is not in the triple database."*

```
Found 1208 distinct paths, 48 after minfreq filtering.
Found 1506 path instances, 275 after minfreq filtering.

MOST SIMILAR RULES FOR: bought
1. bought                         1
2. agreed in principle to acquire 0.819795881096714
3. may buy                        0.66944148791469
4. has acquired                   0.666903852816526
5. raised                         0.656327022371491

MOST SIMILAR RULES FOR: may buy
1. may buy                        1
2. agreed to buy                  0.630805299801703
3. agreed in principle to acquire 0.618502614909011
4. has acquired                   0.510126583609066
5. sold                           0.280441521021403

MOST SIMILAR RULES FOR: will sell
This phrase is not in the triple database.
```

Note that the first (most similar) rule should always be the same as the test phrase because the similarity function should return a value of 1 when measuring the similarity of a path with itself. This is a useful check to verify that your similarity function is implemented correctly.

GRADING CRITERIA

Your program will be graded based on new data 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 files.

ELECTRONIC SUBMISSION INSTRUCTIONS
(a.k.a. "What to turn in and how to do it")

Please use CANVAS to submit an archived (.tar) or zipped (.zip) file containing the following:

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

2. A README file that includes the following information:

   - how to compile and run your code
   - which CADE machine you tested your program on
     (this info may be useful to us if we have trouble running your program)
   - any known bugs, problems, or limitations of your program

   REMINDER: your program must be written in Python or Java and it must compile and run on the unix-based CADE machines! We will not grade programs that cannot be run on the unix-based CADE machines.