DALHOUSIE UNIVERSITY
Inspiring Minds

Winter 2019 (Jan7-Apr8)
Faculty of Computer Science

CSCI 4152/6509 — Natural Language Processing

Assignment 2

Due: *Monday, Feb 18, 2019 by midnight*
Worth: 137 marks (= 25 + 22 + 30 + 40 + 20)
Instructor: Vlado Keselj, CS bldg 432, 902.494.2893, vlado@dnlp.ca

---

Assignment Instructions:

The first two questions of Assignment 2 refer to the Labs 3 and 4, so you will need to follow the lab instructions, and submit your files in the directories `lab3` and `lab4` in your main SVN course directory.

For the rest of the Assignment 2 questions, similarly to Assignment 1, the solutions must be submitted using SVN, in the `a2` directory of your main SVN directory. This means that to solve these questions, your first commands after logging in into your bluenose account should probably be:

```
cd csci4152/CSID
```

or

```
cd csci6509/CSID
```

and then

```
svn mkdir a2
cd a2
```

All files must be plain-text files, unless specified differently by the question.

1) (25 marks, files in *CSID*`/lab3` in the SVN repository) Complete the Lab 3 as instructed. In particular, you will need to properly:

a) (5 marks) Submit the file '`matching.pl`' as instructed,

b) (5 marks) Submit the file '`matching-data.pl`' as instructed,

c) (5 marks) Submit the file '`word_counter.pl`' as instructed,

d) (5 marks) Submit the file '`replace.pl`' as instructed, and

e) (5 marks) Submit the file '`ngram-output.txt`' as instructed.

Note that any program that you submit needs to compile. Even if a complete source code is given in the lab, you need to type it instead of using cut-and-paste, and you need to make sure not to introduce any errors into the program. This follows from the lab instructions

that programs must be tested before submitting.

2) (22 marks, files in *CSID*/`lab4`) Complete Lab 4 as instructed. In particular, you will need to properly:

a) (3 marks) Submit the example file '`array-examples.pl`' as instructed (Step 2).

b) (3 marks) Submit the example file '`test-hash.pl`' as instructed (Step 3).

c) (5 marks) Submit the program file '`letter_counter_blanks.pl`' and the output file '`out_letters.txt`' as instructed (Step 4).

d) (6 marks) Submit the program '`word_counter.pl`' and the output file '`out_word_counter.txt`' as instructed (Step 5).

e) (5 marks) Submit the program '`word_counter2.pl`' as instructed (Step 6).

Note that any program that you submit needs to compile. Even if a complete source code is given in the lab, you need to type it instead of using cut-and-paste, and you need to make sure not to introduce any errors into the program. This follows from the lab instructions that programs must be tested before submitting.

3) (30 marks, `a2q3.txt` or `a2q3.pdf`) In this question, you are asked to do some calculations and include them either in a textual file called `a2q3.txt` or a PDF file called `a2q3.pdf`, which must be submitted in the directory `a2` of your SVN course repository.

Let us assume that five very simple documents are given as follows:

```
d1: the dog ate the homework
d2: the cat ate the homework
d3: the dog and the cat ate the hot dog
d4: the dog and the cat wrote the homework
d5: yesterday was a hot day
```

The names `d1`, `d2`, etc. are names of the documents, and each document is simply the line of text following the name and the colon. So, the first document `d1` contains just the text:
`the dog ate the homework`
Note that the terms (words) in the documents are stemmed, so they may not be always gramatically correct.

a) (20 marks) Your first task is to calculate the document vectors using the *tfidf* weights, according to the vector space model discussed in the class. When presenting vectors, you should assume that stopwords 'the', 'and', 'was', and 'a' are removed, and you should sort the term in the lexicographic order.

You need to show the intermediate calculation results, in particular, the *df* and *idf* for all terms, and also *tf* and *idf* values for all terms over all documents. At the end, show

clearly what are the document vectors for all five documents. For simplicity reasons, you can show all calculations with precision of only two decimal places. For the calculation of log (logarithm) function, you should assume the natural logaritm (base e).

b) (10 marks) Calculate cosine similarity between vectors for documents `d1` and `d2`, and then also between vectors for documents `d1` and `d5`. Is `d1` more similar to `d2` or `d5` according to the cosine similarity measure?

Show the intermediate calculation. You can use precision of just two decimal places, or full precision since there are not that many numbers.

4) (40 marks, SVN submit file: *CSID*`/a2/a2q4.txt` or *CSID*`/a2/a2q4.pdf`) **Note:** You can submit this solution as a plain-text or a PDF file if you prefer.

Let us assume that we are running an experiment in authorship attribution, and our task is to classify journal articles into three different categories: A, B, and C (for three different authors A, B, and C). We obtained the following results:

Gold standard	Classifier output	Number of documents
A	A	18
A	B	7
A	C	10
B	A	18
B	B	42
B	C	17
C	A	16
C	B	19
C	C	33

a) (5 marks) Present the results in a confusion matrix form?

b) (5 marks) What is the accuracy of the classifier?

c) (10 marks) What are precision, recall, and F-measure ($\beta = 1$) for the class 'A'?

d) (10 marks) What are the overall precision, recall, and F-measure with macro-averaging?

e) (10 marks) What are the overall precision, recall, and F-measure with micro-averaging?

Note: In assignments, always include intermediate results and sufficient details about the way the results are obtained.

5) (20 marks, SVN submit file: *CSID*`/a2/a2q5.pl`) Your task is to implement a text tokenizer according to the specifications in this question. The program can be implemented in Perl and must be named `a2q5.pl`, or you can use Python and submit file named `a2q5.py`, Java in file named `a2q5.java`, C in a file named `a2q5.java`, or C++ in a a file named `a2q5.cc`. Other programming languages may be allowed—you would need to check with the instructor. If you use Python, you must identify the Python version in the comment of the

program.

The program must read the standard input and write to the standard output, must run on the bluenose server, and must match the given input and output that will be provided. The program should read input line by line and print each line tokenized. The tokens must be separated by a single space and there should be no spaces at the begining and ending of each line.

The rules for tokenization are as follows:

Rule 1: First, the tokens should be separated based on whitespace characters between them,

Rule 2: The period at the end of a token should be separated as a separate token (e.g., `said.` should be separated as `said` and `.`), except if the period follows the following regular expression:

```
(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|
Mr|Mrs|Miss|Dr|Inc|Corp|
[A-Z] | [A-Z]\.[A-Z])
```

(ignore spaces and newlines in the above regular expression). For example, a period in the following examples should not be separated from the tokens: `Jan.` `Mrs.` `D.` or `B.C`.

Rule 3: Any of the strings described with the regular expression `/([,"]|'')/` at the end of token should be separated from the token into a separate token.

Rule 4: If a token contains a word character (regular expression `/\w/`), then a quote (`'`), followed by possibly some word characters at the end (`/\w*/`), then the quote and the remaining characters should be separated into a separate token; e.g.: `he's` → `he` + `'s` or `Smiths'` → `Smith` + `'`

Rule 5: Two backquotes (``) at the beginning of a token should be separated from the token into a separate token.

Rule 6: If there is a sequence of two or more minus characters (regular expression `/--+/`) inside a token, then the token should be broken around these sequence. If the sequence is at the very beginning or end of a token, it should be separated as well.

Rule 7: The above rules should be applied as many times as needed until tokens are not broken any more. For example, the token `said.,''` should at the end be broken into tokens: `said . , ''`

These rules are created in order to follow tokenization based on an example from the known WSJ (Wall Street Journal) corpus used for part-of-speech tagging. Two sample input

and output files will be provided in the assignment web directory and in a public directory on bluenose.

Notes about coding: Your solution file must start with a header comment like this:

```
#!/usr/bin/perl
# Course: CSCI 4152/6509
# Author: Vlado Keselj
# Description: Solution a2q5.pl
```

or similar for other programming languages.