Lab 4, Working With Text Documents

**Due date:** Wednesday, April 15, 11:59am

**Note:** Due to the postponement of the faculty strike, we are now able to extend the deadline for this lab and to make it somewhat more meaningful in the context of this course. Lab 5, which was advertised to you earlier as a *mini project* will be assigned during the April 13 class.

## Lab Assignment

### Assignment Preparation

This is an individual lab.

The lab involves working with text documents and performing a number of tasks. Due to our ability to extend the lab deadline, you will be asked to perform a(n almost) complete round of data science process. Specifically, you will be dealing with the following steps:

- Data Collection. This step is going to be largely simple, as the data is going to be provided to you in a striaghtforward way and in plain text format.
- Data Cleaning. You will perform tokenization and stopword removal. A simple list of stopwords will be provided.
- Data Modeling. You will build vector space model representations of the documents provided to you. Additionally, you will build some vector space models of subsets of documents provided to you (see instructions below).
- Data Analysis. You will perform simple data analysis on the document representations you have constructed.
- Data Visualization. Some of the analysis you perform will lend itself nicely to a data visualization exercise. You will perform this exercise manually.

### The Data Collection: Reuter 50-50 Datasets

Reuter 50-50 collection of text documents[1] is a well-known and well-studied dataset often used in machine learning and information retrieval coursework, as well as research.

The dataset consists of a collection of news stories published by the Reuters news agencies. The dataset is broken into two parts called training set and test set. For our lab, the intents of these two parts of the dataset are not important, but we will utilize this breakdown of available data.

The dataset was constructed to study machine learning algorithms for authorship attribution. It consists of a selection of 50 authors who published news stories with Reuters. For each author, exactly 100 news stories they authored is placed in the dataset. 50 of the stories are placed in the training set part of the dataset and the remaining 50 – in the test set.

The dataset is available both from Lab 4 course web page, as well as from the University of California at Irvine (UCI) Machine Learning Datasets repository as a single zip file named `C50.zip`. When the file is unzipped, it creates the following directory structure:

```
dekhtyar@csclnx11:~/.../C50 $ ls -al
total 8056
drwx------.  4 dekhtyar domain^users    4096 Apr  7 23:50 .
drwx------.  5 dekhtyar domain^users    4096 Apr  7 23:48 ..
drwx------. 52 dekhtyar domain^users    4096 Apr  7 23:50 C50test
drwx------. 52 dekhtyar domain^users    4096 Apr  7 23:50 C50train
-rw-------.  1 dekhtyar domain^users 8194031 Apr  7 22:15 C50.zip

dekhtyar@csclnx11:~/.../C50 $ ls C50test/
AaronPressman     JaneMacartney     LydiaZajc         RobinSidel
AlanCrosby        JanLopatka        LynneO'Donnell    RogerFillion
AlexanderSmith    JimGilchrist      LynnleyBrowning   SamuelPerry
BenjaminKangLim   JoeOrtiz          MarcelMichelson   SarahDavison
BernardHickey     JohnMastrini      MarkBendeich      ScottHillis
BradDorfman       JonathanBirt      MartinWolk        SimonCowell
DarrenSchuettler  JoWinterbottom    MatthewBunce      TanEeLyn
DavidLawder       KarlPenhaul       MichaelConnor     TheresePoletti
EdnaFernandes     KeithWeir         MureDickie        TimFarrand
EricAuchard       KevinDrawbaugh    NickLouth         ToddNissen
FumikoFujisaki    KevinMorrison     PatriciaCommins   WilliamKazer
GrahamEarnshaw    KirstinRidley     PeterHumphrey
HeatherScoffield  KouroshKarimkhany PierreTran
```

The `C50train` directory has exactly the same structure.

Each directory inside `C50train` and `C50test` directories bears the name of one of the 50 authors. Inside each directory is 50 `.txt` files containing the stories written by that author. Each story is in a separate file whose name follows the following pattern:

[1] https://archive.ics.uci.edu/ml/datasets/Reuter_50_50

```
<number>newsML.txt
```

where `<number>` is a 5- or 6-digit number representing the unique ID of the story[2].

## The Task

### Overview

As data scientists we are insterested in studying the following aspects of the Reuter 50-50 dataset.

**Aspect 1. Document representation.** We want to build the vector space model representations of the news stories of this dataset based on the tf-idf keyword weighting scheme. This, by itself, is not a data analytical task, but proper modeling of the text documents allows for further analysis to proceed.

**Aspect 2. Modeling authorship.** We are interested in understadning of the specifics each author's writing. One of the ways in which we can represent these specifics is to construct a vector space model representation of each author's vocabulary. We will do it by combining all 50 stories written by each author and stored in each part of the dataset into a single "virtual" document and by building a vector of tf-idf keyword weights for each use document.

**Aspect 3. Document comparison.** We want to compare the documents written by each author and by different authors using the cosine similarity measure. We want to understand whether the articles written by the same author tend to be more similar than the articles written by different authors.

**Aspect 4. Author comparison.** We want to understand how similar the collections of articles by the same author from test set and training set are. We can use the cosine similarity measure for this purpose as well.

**Aspect 5. Word Clouds.** Finally, we are interested in visual representations of different authors. We want to build expressive word clouds (tag clouds) summarizing the stories written by each of the authors you are studying.

[2] The Reuter 50-50 dataset was built out of a much larger data set of Reuters stories, so there is no rhyme or reason to the numbers contained in the filenames. All you need to know is that they are all unique.

**Data to work with**

Each of you will receive a list of **five** authors, whose work they will study. You will work with the subset of the overall dataset consisting of the five directories belonging to these authors from the training set and their five directories from the test set. You can ignore, or even delete all other files from the Reuter 50-50 dataset - you will not need them for this lab. Each person will receive a different set of authors chosen essentially at random.

To complete the study, you must perform the following work with your files.

## Build Everything From Scratch.

For this assignment you are expected to build all code from scratch. You can use standard Python tools for accessing files in the directory, and you can read the files line-by-line and use standard string processing functions to split the strings and find specific content in them.

**Do not use someone else's text file parsers.** In fact, **all code for this lab must be written by you.**

## Step 1. Parsing and tokenization.

Write a simple text tokenizer that, given a file name, outputs a list of labelled "tokens" representing individual words, punctuation, numbers and other objects found in the text.

The tokenizer shall operate as follows.

**Types of tokens.** Use four token types:

1. Word. Basically, assigned to any group of letters uninterrupted by punctuation (with the exception of the apostrophe character which requires a bit of special handling described below).
2. Punctuation. Standard punctuation characters:

   `. , ? ! @ # $ % ^ & * ( ) [ ] { } - _ = + ' " : ; / \`

   With the exception of "$", "%", ".", "," and "-" , which may also appear in numbers and ; all other characters from the list above shall always be treated as punctuation.
3. Numbers. Numbers as sequences of digits, as well as the following:
   - a sequence of digits followed by a "." followed by another sequence of digits.

- a sequence of digits followed by a ”,” followed by another sequence of digits, followed by a ”,” multiple times, possibly ending with a ”.” followed by another sequence of digits (very large numbers).
- any of the above preceded by a ”$” sign.
- any of the above preceded by a ”-” (minus) sign.
- any of the above succeeded by a ”%” character (with no spaces between them).

You can determine if someone is a number by checking if it can be converted to an integer or floating point (with or without the front ”$” and/or trailing ”%”).

4. Other. Anything that is not any of the three types above.

**Whitespace.** Generally speaking, whitespace (spaces, tabs, newlines and line feeds) serves as delimiter between tokens, but is not tokenized itself. However, due to the fact that in English there is not space between a word and some punctuation characters that follow, not all tokens are separated by whitespace.

**Special case handling.** If a character ís located between two letters, treat it as an apostrophe and do not break the word with it. For example, `"John's"` shall be treated as a single token. If at least one character surrounding the character ís NOT a letter, treat ás a single quote punctuation token.

**Representation.** The cleanest way to represent a token is to define a Python object `token` that has two attributes: `tokentype` and `value`. However, Python allows for other ways of representing tokens as well. For example, you can represent each token as a tuple

```
(tokenType, content)
```

or even as a two-element list

```
[tokenType, content]
```

Any way that enables you to process the tokens properly in the code that follows will work.

**Function spec.** For this lab, I am not going to provide exact specs – you can choose what is convenient for you, but a simple tokenizer can be built on the basis of two functions that are defined as follows:

```
# tokenizer() : turns a single file into a list of tokens
# parameter: filename (string) - name of/path to the file to be parsed
```

```
def tokenizer(filename) :

# put your code here


# getToken(chunk) takes as input a single chunk of text and returns back a token that represents it
#                  (or tokens if more than one token is needed)

def getToken(chunk):


# put your code here
```

### Step 2: Cleanup and Stopword Removal

Write two functions:

1. `tokenCleanup(tokenList)`. This function takes as input the token list produced by your tokenizer function and cleans it, by removing from the list all non-word tokens[3]. The output of the function is the list of word tokens from the input token list.

2. `removeStopwords(tokenList, stopWordFilename)`. This function takes as input two arguments. The first is the list of tokens (usually, produced by the `tokenCleanup()` function). The second argument is the name of/path to the file containing the list of stopwords. `removeStopwords()` checks, for each *word* token in the input token list if its content is on the liat of stopwords. If it is, the token is removed from the list. Otherwise, the token is placed in the output list. The output of this function is the list of *non-stopword* tokens from the input token list.

**Stopword file.** A stopword file is provided to you on the Lab 4 web page. The stopword file has the following format: it is a plain text file containing a list of stopwords, one stopword per line.

### Step 3: Vector Space Models of Documents and Authors

For each text document you are working with build its vector space model representation with tf-idf keyword weights. When building the model, please take into account the following.

**Vocabulary.** The overall vocabulary of the data collection you are working with is the *set of all non-stopword words* found in all 500 documents assigned

[3]We could have cleaned up the token list in the tokenizer itself, but to make your tokenizer more versatile - in some cases punctuations and numbers do matter! - we elect to have a separate cleanup function.

to you for processing. You must process all 500 documents before computing the document frequencies of each word in the vocabulary.

This assures that vector representations of all documents are built in the same N-dimensional space.

**Term frequency.** Given the general sizes of the documents, we cap the *maximum meaningful term frequency* at 5. That is, given a document, and a keyword $t_i$, if $x$ is the number of occurrences of $t$ in the document, the term frequency $tf_i$ of $t_i$ in the document is computed as follows:

$$tf_i = \begin{cases} \frac{x}{5} & \text{if } x < 5 \\ 1 & \text{otherwise} \end{cases}$$

**Processing.** When processing each file, create the vector (list) of keyword term frequencies for it. *Additionally*, maintain the current word list/vocabulary and the two frequency counts associated with each word:

- overall frequency of occurences of the word in the document collection you are processing, and
- document frequency, i.e., the number of documents that contain the keyword.

**Document vectors.** Python's dictionaries are convenient means for storing both document vectors, *and* the vocabulary, although, vocabulary can also be stored in a simple list - recall that once you assign each keyword an index number, this number cannot be changed.

**Traversing through documents.** You can hardcode your traversal through the Reuter 50-50 directory structure and limit it to the specific directory names you are given. However, **you must traverse the exact directory structure** of the the Reuter 50-50 dataset.

**Persisting tf-idf vectors.** The output of the data modeling process shall be a single data file containing the tf-idf vectors for each of the documents. You are welcome to choose the format in which you are going to store the vectors. The two key requirements to your format are:

- Each vector must start with the filename for the document you are representing. The filename must contain a complete path to the file from the root of the Reuter 50-50 dataset: it should look like something like this: `C50test/AlanCrosby/22649newsML.txt`. After the filename, you can print out the vector of keyword weights in any form that you find convenient.

- You must be able to properly read the vector of tf-idf weights from your output file. The analytical part of your lab assignment will take the data file containing the document vectors as input.

**Step 3.5: Building Vectors of Keyword Weights for Authors**

In addition to building the vectors of keyword weights for individual documents in your collection, you will build 10 vectors of keyword weights for the directories in the collection you are processing. Each author will have two vectors associated with them: one for the documents in the training set, and one - for the documents in the test set.

**Term Frequencies.** To build a vector of keyword weights for each author's directory, assume that the combined content of all 50 articles in the diectory constitutes a single "document". For this document, term frequency shall be computed by

- first, discovering the most frequent term found in the 50 articles, and its frequency $maxF$;
- and second, computing the term frequency $tf_i$ of any term $t_i$ which occurs a combined $n$ times in the 50 articles as

$$tf_i = \frac{n}{maxF}$$

**Note:** the term frequency computation makes the tf-idf vectors computed for authors a bit imcompatible with the tf-idf vectors computed for individual documents. This is ok though, as in this lab we will not be comparing author vectors to individual article vectors.

**Inverse Document Frequencies.** Given the notion of the "document" representing the author, the document frequency of a word is the number of directories in your subset of the Reuter 50-50 dataset, in which the word occurs in at least one article.

**Persisting tf-idf vectors.** Follow similar ideas as for persisting the tf-idf vectors for articles. Specifically, the output of your process of modeling authors shall be a single data file containing the 10 author vectors you built. Each vector shall be named after the path to the directory from the root of the dataset (e.g., `C50test/AlanCrosby`). The representations of the vectors in the file between the author vectors and the article vectors shall be consistent (this makes your life easy - you can write just one set of functions to dump vectors into a file).

### Step 4: Document Comparison

This is the first analytical task you will perform. Please note, all analytical tasks **must read the data** from the data files containing the tf-idf vectors that you build on previous step.

Your goal for this task is to compare document representations to each other. To do this, first we need to know how to compare documents.

**Cosine Similarity.** To compare two documents to each other you will compute the cosine similarity measure for their vectors. Given two vectors of keyword weights $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ the cosine similarity between $x$ and $y$ is computed as follows:

$$sim(x, y) = cos(x, y) = \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

**Comparing documents.** Your goal is to determine whether for each author, their articles are more similar to other articles written by them or articles written by other authors. To achieve this goal, you will conduct a guided comparison of articles to each other using the cosine similarity in the following manner:

1. Single Author comparisons. For each directory, find the cosine similarity between each pair of articles in it. Compute, **and retain** the following values:
   - The largest similarity between two articles (simMax(Directory))
   - The smallest similarity between two articles (simMin(Directory))
   - The average similarity between two articles (simAvg(Directory))
   - The standard deviation of all similarities between a pair of different articles from the directory (stdDev(Directory))
2. Single Author comparisons between subsets. For each pair of directories from the test set and training set *representing the work of the same author*, compute the cosine similarities between every pair of articles where one articles is from the training set directory and the other - from the test set directory. Compute, and retain the following values:
   - The largest similarity between two compared articles (simMax(Author))
   - The smallest similarity between two compared articles (simMin(Author))
   - The average similarity between two compared articles (simAvg(Author))
   - The standard deviation of all similarities between a pair of compared articles (stdDev(Author))

3. Comparison of different authors. The following needs to be performed **separately** for the training set and test set (i.e., only compare directories where both directories are either in the training set or test set. Do not perform mixed comparisons.)

   For each pair of directories within the training set (test set) compute the cosine similarity. Compute and retain the following values:

   - The largest similarity between two articles (simMax(Directory1, Directory2))
   - The smallest similarity between two articles (simMin(Directory1, Directory2))
   - The average similarity between two articles (simAvg(Directory1, Directory2))
   - The standard deviation of all similarities between a pair of different articles from the directory (stdDev(Directory1,Directory2))

   If you compute for the pair (Directory1, Directory2), there is no need to duplicate computations for (Directory2, Directory1) pair.

4. Store computed results. Store all computed values in a single output file in a format that is both easily machine-readable (i.e., you should be able to write a piece of code that reads the values back in and undestands the meaning of each of them) and himan-readable.

   **Note:** 2D matrices with rows and columns representing either individual directories, or individual authrs and cells representing specific values may be a convenient presentation/storage format for at least some of the date computed above.

## Step 5: Author Analysis

Perform analysis of the similarity between the author vectors similarly to the analysis you performed on Step 4. Specifically, perform the following sets of comparisons:

1. Same author, different subsets. For each author, find the similarity between their training set and test set articles by computing the cosine similarity between their training set and test set author vectors. Output the cosine similarities you obtain.

2. Different authors, same subset. Perform two sets of cosine similarity computations between author vectors for different authors: one for all pairs of author vectors for the training set and one – for all pairs of author vectors for the test set. Output the cosine similarities you obtain.

3. Output. Output all results obtained for this step in a single output file that is both human and machine-readable.

## Step 6: Word Clouds

Perform the following tasks

- Select the top 40 most frequent words for each directory complete with the raw counts. Output these words into a single file in the following format (for each word):

  `<Freq> <Word>`

  Here, `<Freq>` is the raw frequency (the total number of occurrences) of a word in all the articles in the directory, and `<Word>` is the word itself.

- Additionally, do the same *for each author* - i.e., combine the frequency counts for keywords from the two directories for each author into a single word frequency list and output the top 40 most frequent words from it in the same format as above.

- Create a word cloud for each of the 15 frequency lists obtained in the steps above.

  Use http://www.wordclouds.com/ as your word cloud provider.

  Use the Word list data upload feature.

  Edit each word cloud for size (make sure all words appear), shape, color scheme, font, and any other editable parameters.

  Create a .png file of the version of the word cloud you like the most.

## Step 7: Reflection

This lab has asked you to do a lot of things. Let's summarize them now.

Write a concise report analyzing the data you received on Steps 4 and 5 and 6. Answer the following questions to the best of your ability based on the data you observed:

1. How diverse is the writing of each individual author in your subset of the Reuter 50-50 dataset?
2. How similar/different do the articles written by the same author by stored in differents parts of the data set appear to be (i.e., are the articles from the training set written by one author similar to the same author's articles from the test set.
3. How similar/different are the writings of different authors in your subset of the Reuter 50-50 dataset?
4. Can you find an easy way to characterize the writings of each of the authors in your subset of the Reuter 50-50 dataset based **solely on the outputs of** Steps 4-6? How would you characterize their writings in terms of main themes?

### Submission

For this lab we will use CSL `handin` tool.

You shall submit the following.

- All the code you wrote to fulfill the requirements of this lab.
- A `README` file with instructions on how to run your code to perform different tasks of the assignment.
- A `AUTHORS` files which simply contains the list of the authors you studied in the lab.
- Your report in PDF format.
- The word cloud files you built. The files shall be named `C50test-Author.png`, `C50train-Author.png` and `Author.png` for `test set`, `training set` and combined data, with `Autor` replaced by the directory name matching each of the authors. Place all PNG files in the `wordcloud` directory.

Place **all** files and directories *except* for the report file, into a single archive named `lab04.tar.gz` or `lab04.zip` and submit it. Submit the PDF version of the report separately, outside of the archive.

Use `handin` to submit as follows:

```
$ handin dekhtyar lab04 <FILES>
```

**Good Luck!**