Lab 4, Information Retrieval and Text Mining

Due date: Thursday, May 24, 11:59pm

Note: I am assigning this lab a bit early, because we do not have a May 15 lecture. Start working on this lab **after** you complete Lab 3.

Lab Assignment

This is a pair programming lab.

As part of this assignment you are asked to analyze a collection of text documents, the Reuter 50-50 dataset that is often used in text mining and text classification tasks. You will create vector space representations of the documents from this dataset, and will use these representations to conduct two studies:

- **Study 1:** Determine if the K-Nearest Neighbors algorithm can be used to reliably establish the authorship of the text documents in the dataset.
- **Study 2:** Determine if Agglomerative Hierarchical Clustering algorithm can be used to group the documents by authors (i.e., determine the authorship purity of clusters constructed by the algorithm.

Notice that you get to reuse some of the Lab 3 code in this assignment.

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.

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

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 and we will use both parts of the dataset together.

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 exactily 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:

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

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

[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.

The Task

Preliminary Steps

The first task for you assignment is to create vectorized tf-idf representations of each document and to store these representations.

Write a program `textVectorizer.java` or `textVectorizer.py`[3] that takes as input the location of the root of the Reurters 50-50 dataset directory, and produces, as the result, a file containing the vectorized tf-idf representations for each of the 5000 documents in the dataset. The name of the output file can be an input parameter to your program (for the sake of flexibility). Additionally, your program shall produce (to make your life easier during the evaluation stages) a ground truth file that for each document (identified by its file name) stores the name of the author (name of the directory the file is in). The ground truth file can have any format you want, but a simple CSV file looking as follows:

```
421829newsML.txt,AaronPressman
424074newsML.txt,AaronPressman
...
236352newsML.txt,AlanCrosby
293241newsML.txt,AlanCrosby
...
```

Note: You must write the entire vectorizer from scratch without the use of text vectorization tools/APIs/functionality available in the specialized packages in the programming language of your choice. You are **explicitly prohibited** from using any `scikit learn` or `nltk` Python package functionality for this lab (as well as their counterparts in other languages). The point of this assignment is to learn **how to build** this functionality.

You can use standard parsing techniques available to you in your programming languages such as `split()` and `strip()` methods. You can take advantage of the data structures provided to you by the `NumPy` package (or similar packages in other programming languages).

When implementing vectorized representations of the documents, please observe that the overall vocabulary of the Reuter 50-50 dataset is significantly larger than the number of unique words used in any specific single document, and therefore, the tf-idf vectors of individual documents will be rather sparse.

To support your vectorization efforts, and other tasks of this assignment, you will implement a `Vector` class (or a `Vector` ADT) in your host programming language. Your `Vector` class will store sparse tf-idf vector repre-

[3] As usual, if you are using a different programming language, name your program accordingly.

sentations of text documents. It shall also implement at least two similarity score computations for a pair of vectors: cosine similarity and okapi.

Authorship Attribution Through KNN

The first analytical task for this assignment is to implement the K-Nearest Neighbors classifier and use it to test the accuracy of detecting each of the authors.

K-Nearest Neighbors requires a distance or similarity metric to properly operate. You will use the similarity metrics implemented on the previous stage: cosine and okapi in this exercise.

Write a program `knnAuthorship.java/knnAuthorship.py` which takes as input the file of vectorized document representations, a value of k (number of nearest neighbors to check), and a flag indicating the similarity metric to be used in the computation[4] The output of the program shall be and authorship label predicted for each of the documents in the Reuters 50-50 dataset. Essentially, your KNN implementation shall use an *all-but-one* validation (take a document, find k nearest neighbors among the remaining 4999 documents in the dataset). The output can be printed out directly to the screen, or placed in an output file.

To evaluate the accuracy of the predictions, write an `classifierEvaluation.java` or `classfierEvaluation.py` program that takes as input the file generated by the `knnAuthorship` program, as well as the ground truth file (remember you had to generate one!)[5], and produces as the result the following output:

- For each author in the dataset, output the total number of *hits* (correctly predicted documents), *strikes* (false positives predicted) and *misses* (document written by the author, which were not attributed to the author).
- For each author in the dataset report the precision, the recall, and the f1-measure of the KNN procedure.
- Overall, report the total number of documents with correctly predicted authors, the total number of documents with incorrectly predicted authors, and the overall accuracy of the prediction.
- The full 50x50 confusion matrix in a CSV form (with the top row, and first column containing the author names/labels). This can be dumped directly into an output file, rather than printed to screen.

Your analytical goal is to determine which authors are easier to predict authorship for, and which authors are hard (and who they tend to get confused with). You can do the final analytical step by hand - simply looking at the results of your evaluation program, or, alternatively, you can add code to your evaluation program that reports this information out automatically.

[4]If needed, your program can also take any other well-documented input parameters.
[5]It may be helpful if both files are stored in exactly the same format.

Authorship Attribution Through Clustering

Your second analysis comes in a form of clustering the vectorized representations of the Reuters 50-50 documents. For this project is it convenient to use hierarchical agglomerative clustering. While K-means also allows for clustering for a given number of clusters (50, in our case), hierarchical clustering method is more flexible in that not only does it allow for building of 50 clusters, but if some of these clusters are degenerate (e.g., one of the 50 cluster is a single outlier document), you may be able to construct a larger set of clusters from the same dendrogram.

You are allowed - and in fact - encouraged to reuse your Lab 3 implementation of the hierarchical clustering algorithm. Recall that the actual clustering procedure should take as input the distance matrix, so you should be able to reuse significant portions of your code without having to deal with different data formats. The almost week-long overlap in available instructions for Lab 3 and Lab 4 should provide you a good incentive to plan ahead.

For this analytical tasks you will write the following programs. First, build a program `clusteringAuthorship.java`/`clusteringAuthorship.py`, which takes as input a the file of vectorized document representations, and any parameters for setting up your hierarchical clustering implementation (e.g., the linkage). Just as your Lab 3 implementation, this program shall output (directly to file, please! it will be very large!) the XML or JSON version of the dendrogram representing the clustering.

Your second program, `clusterEvaluation.java`/`clusterEvaluation.py` shall take as input a dendrogram generated by `clusteringAuthorship` and shall analyze it as follows:

- First, it shall attempt to construct 50 *meaningful* clusters based on the input dendrogram. In most cases, this should result in simply undoing the last 50 mergers, however, please be mindful, that some clusters that you may get this way may be way too small to be considered an appropriate representation of a single author. You may want to continue splitting into clusters until you have 50 clusters of "reasonable" sizes. Treat all small-size clusters as essential outliers.
- Second, for the list of clusters you come up with, determine the cluster purity scores (for each cluster), i.e., the percent of the cluster that has the cluster plurality author label. For each cluster, identify the plurality class. Sort the authors by how pure the cluster/clusters that are assigned to them are.
- Third, report the accuracy of detecting each author. A document is a *hit* if it belongs to a cluster where its author is a plurality label. Otherwise it is a *miss*. Compute for each author the precision and the recall (note that if an author is a plurality label in more than one cluster, you will need to combine your hits and misses from multiple clusters).

- Finally, report the overall purity of clusters/ accuracy of detecting authors.

Tuning Parameters

When experimenting with both analytical methods you may wind up running your analyses multiple time with a different set of parameters. For both methods, you can construct different datasets based on what pre-processing techniques were used to create them. Generally speaking, you have four key choices to consider:

- **No stopword removal**, **no stemming**.
- **Stop word removal**, but **no stemming**.
- **Stemming**, but **no stopword removal**.
- both **Stemming** and **stopword removal**.

Within this set of possibilities, different lists of stop words can be used to create more possible inputs to both the classification and clusering authorship attribution methods.

For the KNN-based authorship attribution, you have an additional hyperparameter to tune: K, the number of neighbors to use. You have some natural boundaries on K as part of your procedure: clearly, $K \leq 49$, and quite possibly needs to be made significantly smaller.

For the clustering-based authorship attribution, the key hyperparameter is your choice of linkage, with *single*, *complete* and *average* as viable choices **at a minimum**. (Note that your Lab 3 implementation should also have a parameter specifying whether the dataset needs to be normalized/standardized prior to processing, but due to the construction of the tf-idf vectors, this parameter is not necessary in this lab).

Reporting your findings

Prepare a report that describes the following:

- The exepriment you ran for each of the two analytical methods for determining authorship: what datasets were explored, which values of hyperparameters were investigated.
- The results of your best runs for each of the two analytical methods. You should present in tabular (or easy to read text) form the accuracy measures for determining each author, as well as the overall accuracy.
- A reflection on each of the methods w.r.t. the method's overall ability to properly attribute authorship of the articles, as well as any specific information that stands out: which authors are easy to predict? which are hard? which tend to be confused with each other?

- A comparison of two methods to each other and a final determination of which method proved to be more accurate.

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.
- Output files generated by your evaluation programs for both analytical methods for authorship detection (specify filenames in README file).
- Your report in PDF format.

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 lab04-466 <FILES>
```

Good Luck!