COSC2820 Advanced Programming for Data Science

COSC 2820/2815 Assignment 2: NLP Web-based Data Application Milestone I: Natural Language Processing

1. Overview

Nowadays there are many job hunting websites including seek.com.au and au.indeed.com. These job hunting sites all manage a job search system, where job hunters could search for relevant jobs based on keywords, salary, and categories. In previous years, the category of an advertised job was often manually entered by the advertiser (e.g., the employer). There were mistakes made for category assignment. As a result, the jobs in the wrong class did not get enough exposure to relevant candidate groups.

With advances in text analysis, automated job classification has become feasible; and sensible suggestions for job categories can then be made to potential advertisers. This can help reduce human data entry error, increase the job exposure to relevant candidates, and also improve the user experience of the job hunting site. In order to do so, we need an automated job ads classification system that helps to predict the categories of newly entered job advertisements.

This assessment includes two milestones. The first milestone (NLP) concerns the pipeline from basic text preprocessing to building text classification models for predicting the category of a given job advertisement. Then, the second milestone will adopt one of the models that we built in the first milestone, and develop a job hunting website that allows users to browse existing job advertisements, as well as for employers to create new job advertisements.
This assessment description is about Milestone 1: Natural Language Processing.

2. Learning Outcomes

This assessment relates to following learning outcomes of the course:

- CLO 4: Pre-process natural language text data to generate effective feature representations;
- CLO 5: Document and maintain an editable transcript of the data pre-processing pipeline for professional reporting.

3. Assessment details

In this milestone, you are required to pre-process a collection of job advertisement documents, build machine learning models for document classification (i.e., classifying the category of a given job advertisement), and perform evaluation and analysis on the built models.

The Data

In this assignment, you are given a small collection of job advertisement documents (around 750 jobs). The data folder is available for download from canvas.
Inside the data folder you will see 4 different subfolders, namely: Accounting_Finance, Engineering, Healthcare_Nursing, and Sales, **each folder name is a job category**.
The job advertisement text documents of a particular category are located in the corresponding subfolder.
Each job advertisement document is a txt file, named as "Job_<ID>.txt". It contains the title, the webindex, (some will also have information on the company name, some might not), and the full description of the job advertisement.

Task 1: Basic Text Pre-processing [5 marks]

In this task, you are required to perform basic text pre-processing on the given dataset, including, but not limited to tokenization, removing most/less frequent words and stop words. In this task, we focus on pre-processing the **description** only. You are required to perform the following:

1. Extract information from each job advertisement. Perform the following pre-processing steps to the **description** of each job advertisement;
2. Tokenize each job advertisement description. The word tokenization must use the following regular expression, **r"[a-zA-Z]+(?:[-'][a-zA-Z]+)?"**;
3. All the words must be converted into the lower case;
4. Remove words with length less than 2.
5. Remove stopwords using the provided stop words list (i.e, **stopwords_en.txt**). It is located inside the same downloaded folder.
6. Remove the word that appears only once in the document collection, based on **term** frequency.
7. Remove the top 50 most frequent words based on **document** frequency.
8. Save all job advertisement text and information in txt file(s) (you have flexibility to choose what format you want to save the preprocessed job ads, and you will need to retrieve the pre-processed job ads text in Task 2 & 3);
9. Build a vocabulary of the cleaned job advertisement descriptions, save it in a txt file (please refer to the required output);

Note:

- For all the words that we removed (including step 4,5,6,7), you will also exclude them in the generated vocabulary.
- The output of this task will be checked against the expected output. You should strictly follow the following format requirement.

Required Output for Task 1:

The output of this task must contain the following files:

- **vocab.txt** This file contains the **unigram** vocabulary, one each line, in the following format: ***word_string:word_integer_index***. Very importantly, words in the vocabulary must be sorted in alphabetical order, and the index value starts from 0. This file is the key to interpret the sparse encoding. For instance, in the following example, the word *aaron* is the 20th word (the corresponding integer_index as 19) in the vocabulary (note that the index values and words in the following image are artificial and used to demonstrate the required format only, it doesn't reflect the values of the actual expected output).

```
aand:12
aantrekkelijk:13
aap:14
aar:15
aarca:16
aardman:17
aareon:18
aaron:19
aaronwallis:20
aarosette:21
aarosettehotel:22
```

Fig.1 example format for vocab.txt

- All Python code related to Task 1 should be written in the jupyter notebook **task1.ipynb**.

Task 2: Generating Feature Representations for Job Advertisement Descriptions [10 marks]

In this task, you are required to generate different types of feature representations for the collection of job advertisements. Note that in this task, we will only consider the **description** of the job advertisement. The feature representation that you need to generate includes the following:

Bag-of-words model:

- Generate the **Count** vector representation for each job advertisement description, and save them into a file (please refer to the required output). Note, the generated Count vector representation must be based on the generated **vocabulary** in Task 1 (as saved in **vocab.txt**).

Models based on word embeddings:

- You are required to generate feature representation of job advertisement description based on the following language models, respectively:
  - Choose 1 embedding language model, e.g., FastText, GoogleNews300, or other Word2Vec pretrained models, or Glove.

You are required to build the weighted (i.e., TF-IDF weighted) and unweighted vector representation for each job advertisement description using the chosen language model. To summarize, there are 3 different types of feature representation of documents that you need to build in this task, including count vector, two document embeddings (one TF-IDF weighted, and one unweighted version).

Required Output for Task 2:

- **count_vectors.txt** This file stores the sparse count vector representation of job advertisement descriptions in the following format. Each line of this file corresponds to one advertisement. It starts with a '#' key followed by the webindex of the job advertisement, and a comma ','. The rest of the line is the sparse representation of the corresponding description in the form of ***word_integer_index:word_freq*** separated by comma. Following is an example of the file format (note that the following image is artificial and used to demonstrate the required format only, it doesn't reflect the values of the actual expected output):

```
#12612628,11:1,35:1,255:4,256:1,411:3,503:1,535:2,615:4,727:2,742:1,752:1,782
:1,833:3,862:1,893:1,941:1,1067:1
#12612830,0:1,64:1,70:1,72:1,90:1,92:1,116:1,218:1,231:1,245:1,270:1,274:1,31
0:1,316:1,317:1,330:2,412:2,433:1,441:1,529:1,532:1,537:1,538:1,545:2,616:2,6
17:1,649:1,680:1,697:1,737:1,742:1,744:1,752:1,766:1,768:1,773:2,774:1,799:1,
845:1,855:1,858:1,864:1,874:1,888:2,902:1,937:1,952:2,962:1,1003:1,1008:1,107
7:1,1082:1,1087:1,1092:1,1093:1
#12612844,6:2,17:1,57:1,59:2,134:2,149:2,169:2,196:1,198:1,199:2,208:3,214:1,
218:1,245:1,252:1,255:6,259:1,265:1,327:2,340:1,352:1,387:2,390:2,432:1,450:1
```

Fig. 4 example format for count_vectors.txt

Note: ***word_freq*** here refers to the frequency of the unigram in the corresponding **Description** only, excluding the title.

Task 3: Job Advertisement Classification [10 marks]

In this task, you are required to build machine learning models for classifying the category of a job advertisement text. A simple model that you can consider is the logistic regression model from sklearn as demonstrated in the activities. However, you feel free to select other models (even if it has not been covered in this course). You are required to conduct two sets of experiments on the provided dataset to investigate the following two questions, respectively.

Q1: Language model comparisons

Which language model we built previously (based on job advertisement descriptions) performs the best with the chosen machine learning model? To answer these questions, you are required to build machine learning models based on the feature representations of the documents you generated in task 2, and to perform evaluation on the various model performance.

Q2: Does more information provide higher accuracy?

In Task 2, we have built a number of feature representations of documents based on job advertisement descriptions. However, we have not explored other features of a job advertisement, e.g., the title of the job position. Will adding extra information help to boost up the accuracy of the model? To answer this question, you are required to conduct experiments to build and **compare** the performance of classification models that considering:

- only title of the job advertisement
- only description of the job advertisement (which you've already done in Task 3)
- both title and description of the job advertisement. For this, you have the flexibility to simply concatenate title and description of a job advertisement when generating document feature representation; or to generate separate feature representations for title and description, respectively, and use both features in the classification models.

Note that: For both questions above,

- You can choose one of the language models you experienced in Task 2 to experiment on to answer this question.
- When evaluating the performance of the models, you are required to conduct a **5-fold cross validation** to obtain robust comparisons.

All Python code related to Task 2 and 3 should be written in the jupyter notebook task2_3.ipynb.

6. Marking Guidelines

Marking Criteria

- ***Mechanical pass****:* Your outputs will be compared against the expected output. Therefore, marking will be based on the similarity between what we expect (as discussed in the instructions) and what we receive from you. It is extremely important to carefully follow the instructions to produce the expected output. Otherwise, you may easily lose many points for simple mistakes (e.g. typos in the format of the files, not loading essential libraries, different file names/path, etc).
- ***Expert pass****:* Your jupyter notebook will be checked by an expert to validate the logic and flow, proper use of libraries and functions, and clarity of codes, comments, structure and presentation.

- You need to ensure all the codes and files that are required to run your code are included in the submission. The expert will **NOT** fix your code's problem even if it is a simple typo in a file name or an imported library.

Mark Allocations

- Task 1 Basic Text Pre-processing [5%] o Implementation [4%] o Notebook presentation [1%], proportional to actual mark obtained in implementation
- Task 2 [10%] o Implementation [7%] o Notebook presentation [3%], proportional to actual mark obtained in implementation
- Task 3 [10%] o Implementation [7%]
    - o Notebook presentation [3%], proportional to actual mark obtained in implementation

For Task 1, and Task 2 and 3, you are required to maintain an auditable and editable transcript, and communicate any justification of methods/approach chosen, results, analysis and findings through jupyter notebook. The presentation of the jupyter notebook accounts for certain percentages of the allocated mark for each task, proportional to the actual mark obtained, as per specified above. Students can refer to the activities in modules as examples for the level of details that they should include in their jupyter notebook.