CSE 158/258, Fall 2023: Homework 4

Instructions

Please submit your solution **by Monday Nov 27.** Submissions should be made on **gradescope**. Please complete homework **individually**.

You should submit two files:

`answers_hw4.txt` should contain a python dictionary containing your answers to each question. Its format should be like the following:

```
{ "Q1": 1.5, "Q2": [3,5,17,8], "Q2": "b", (etc.) }
```

The provided code stub demonstrates how to prepare your answers and includes an answer template for each question.

`homework4.py` A python file containing working code for your solutions. The autograder *will not execute your code*; this file is required so that we can assign partial grades in the event of incorrect solutions, check for plagiarism, etc. Your solution should **Clearly document which sections correspond to each question and answer**.

You may build your solution on top of code from the textbook:

Text Mining: `https://cseweb.ucsd.edu/~jmcauley/pml/code/chap8.html`

Content and Structure: `https://cseweb.ucsd.edu/~jmcauley/pml/code/chap6.html`

You will need the following files:

Homework 4 stub : `https://cseweb.ucsd.edu/classes/fa23/cse258-a/stubs/`

Steam Category Data : `https://cseweb.ucsd.edu/classes/fa23/cse258-a/data/steam_category.json.gz`

GoodReads Young Adult Reviews (20,000) : `https://cseweb.ucsd.edu/classes/fa23/cse258-a/data/young_adult_20000.json.gz`

Tasks: Text Mining

1. Using the *Steam* category data, build training/test sets consisting of 10,000 reviews each. Code to do so is provided in the stub.[1] We'll start by building features to represent common words. Start by removing punctuation and capitalization, and finding the 1,000 most common words across all reviews ('text' field) in the training set. See the 'text mining' lectures for code for this process. Report the 10 most common words, along with their frequencies, as a list of (frequency, word) tuples.

2. Build bag-of-words feature vectors by counting the instances of these 1,000 words in each review. Set the labels (y) to be the 'genreID' column for the training instances. You may use these labels directly with sklearn's *LogisticRegression* model, which will automatically perform multiclass classification. Report performance (accuracy) on your test set.

3. What is the *inverse document frequency* of the words 'character', 'game', 'length', 'a', and 'it'? What are their *tf-idf* scores in the first (training) review (using log base 10, unigrams only, following the first definition of tf-idf given in the slides)? *All frequencies etc. should be calculated using the training data only.* (2 marks)

4. Adapt your unigram model to use the tfidf scores of words, rather than a bag-of-words representation. That is, rather than your features containing the word *counts* for the 1000 most common unigrams, it should contain tfidf scores for the 1000 most common unigrams. Report the accuracy of this new model.

[1] Although the data is larger, we'll use only a small fraction for these experiments.

5. Which review *in the test set* the highest cosine similarity compared to the first review in the training set, in terms of their tf-idf representations (considering unigrams only). Provide the cosine similarity score and the reviewID?

6. Try to improve upon the performance of the above classifiers from Questions 2 and 4 by using different dictionary sizes, or changing the regularization constant C passed to the logistic regression model. Report the performance of your solution.

   Use the first half (10,000) of the book review corpus for training and the rest for testing (code to read the data is provided in the stub). Process reviews **without capitalization or punctuation** (and without using stemming or removing stopwords).

These tasks should be completed using the entire dataset of 20,000 reviews from Goodreads:

7. Using the *word2vec* library in *gensim*, fit an item2vec model, treating each 'sentence' as a temporally-ordered[2] list of items per user. Use parameters `min_count=1`, `size=5`, `window=3`, `sg=1`.[3] Report the 5 most similar items to the book from the first review along with their similarity scores (your answer can be the output of the `similar_by_word` function).

[2]You may use `dateutil.parser.parse` to parse the date string.
[3]The `size` argument might be `vector_size` in some library versions.