Assignment 1: Comparison of BOW with Word Embeddings on Text Classification (100 Points)

1 Dates

- **Released**: January 19, 2022
- **Due**: January 25, 2022 11:59pm

2 Introduction

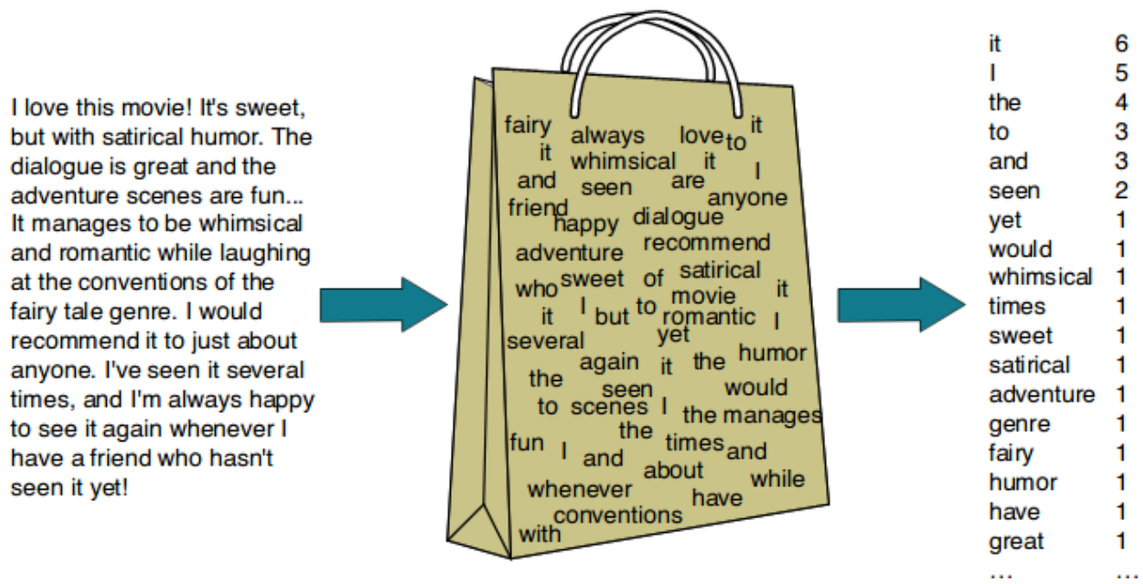
In this assignment, you will examine word representations in a text classification task, implemented in python (using NumPy and PyTorch). Word representations (e.g.word vectors) are often a fundamental component for NLP tasks such as text classification, question answering, summarization, text generation, and so on. It is important to develop intuitions about their strengths and weaknesses. You will explore two types of word vector representations introduced in the class meeting 2, high-dimension BOW (Bag-of-words) representation and low dimension GloVe word embeddings. You will compare them in a text classification task to see which performs better and why.

The assignment materials provide a text dataset, point you to the GloVe vectors, and skeleton code for the functions you will need for loading the data, creating the document representations, and training and testing a classifier. It has existing code for a baseline classifier using simple functions from PyTorch. If you are familiar with machine learning, this part of the code should be understandable. If not, you should make any modifications you can; these kinds of models will be explained later in the class.

3 Text Representation

The point of a data “representation” for machine learning is to convert real-world examples from the problem into a more abstract representation that preserves information relevant to the problem, and is computationally convenient. Here we use vector representations, where the features are the vector positions, and the value of a feature is numeric. The BOW feature vectors are high-dimensional because they are the length of the vocabulary. The GloVe vectors are low-dimensional, ranging from 50 to 300, depending on which GloVe vectors you choose.

3.1 Bag-Of-Words (BOW) Representation



The intuition of the BOW classifier is shown in the above figure. A text is represented as an unordered set of uncased words with a raw or weighted frequency count. (NOTE: BOW vectors where the values at each vector position are absolute counts is the same as a

multiset.) In the figure, word order is ignored and instead (e.g., phrases like “I love this movie” and “I would recommend it”), a vector where each vector position is a word type indicates that the word 'it' occurred 6 times in the entire review, the word 'i' occurred 5 times, and so on.

To create your BOW vectors for the movie review data, you will first find the words in the vocabulary to define the vector length and “meaning” of each vector position (which word the count represents), using words above some frequency level. Then you will convert each review into its BOW vector, to serve as input to the classifier.

3.2 Word Embeddings

Through keeping up with the required course readings, you should be familiar with GloVe as described in the paper by Pennington et al. (2014). To create representations of the movie review texts with GloVe embeddings, you first need to download the pre-trained GloVe vectors from the Stanford page, where you can see four GloVe zip files. USE THE ONE CALLED glove.6b.zip, which was trained on a 2014 snapshot of Wikipedia and Gigaword 5. Do NOT USE any of the others. Unpack the zip file and store the contents in the same folder with the code, in a subfolder called “glove”. The code uses this path, so you must create this location for the 4 glove txt files you find in the zip:

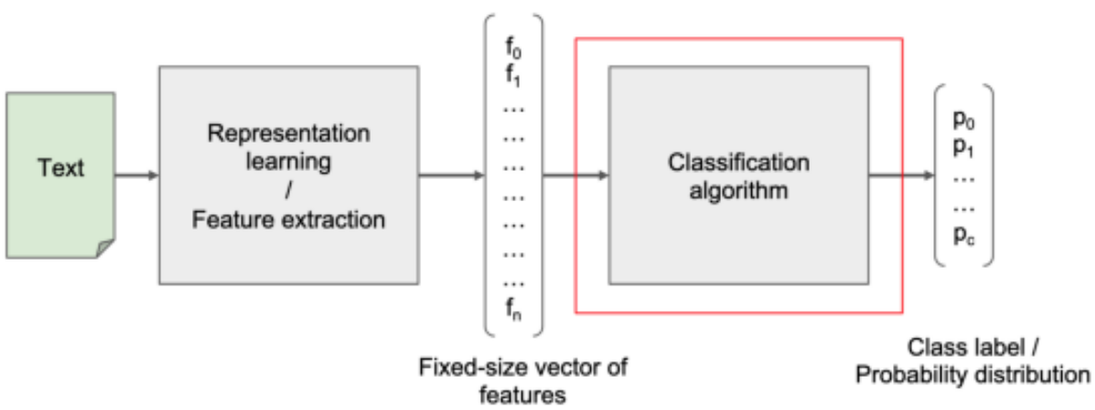
./glove/glove.6B.100d.txt
/glove.6B.200d.txt
/glove.6B.300d.txt
/glove.6B.50d.txt

Each file has 400K rows, where each row starts with a “word” token (sometimes it is a symbol instead of a dictionary word), followed by the glove word vector.

You can use any one of the four glove vector “dictionaries”. Your task is to write a function that converts a movie review into a vector representation. Therefore you need to decide on how you will combine the word vectors. A simple method is to retrieve the glove vectors of length N for every word in a review, and then average them to create a final vector of length N to represent the review. You can do experiments with different size vectors, and different ways of combining them. For example, you could try the weighted averaging discussed in Arora et al., 2017. In the text part of your submission (see section 7), you will need to justify each experiment in terms of a question you were exploring to understand word embeddings and text representation, or in terms of a reasonable hypothesis about the text representation you decided upon.

3.3 Text Classification

The figure below illustrates the text classification pipeline for a fully trained classification algorithm. During training, there is an additional step at the output where the class label is predicted, which is to compare the prediction to the ground truth and use the result in the calculation of the overall loss of the model on a given training epoch. For this assignment, you are using feature extraction, not representation learning.



• **Input**: Some text ***x*** (e.g., sentence, document)
• **Output**: The probability distribution of each label ***y*** over a finite label set
• **Goal**: learn a mapping function ***f*** from ***x*** to ***y***

In this assignment, you will complete and/or modify the existing classifier to test the two kinds of text representation: BOW and a representation based on GloVe word embeddings. The classification task is a binary one for movie review sentiment. Since the amount of data is small, it does not need a very complicated algorithm. The code skeleton suggests you start with a simple three layer neural network as a base. You can modify it or change it to another classifier.

3 Data

The dataset consists of 10K IMDB movie reviews that were prepared for this assignment. In the distribution zip file, you will see a data subfolder with two files: a text file (*review.txt*) and a label file (*label.txt*). Each row in review.txt is a review that has been pre-processed with downcasing, and tokenization. Each row in label.txt is a label on the corresponding review (row) in review.txt. You will conduct experiments on this dataset, and answer questions based on your experiments.

4 Overview of the Code

Two python files are provided: run.py for running the pipeline, and model.py. **Do not modify anything in *run.py*.**

You need to finish or modify some functions in ***model.py:***

def __init__(self,vocab,pos_data,neg_data):
This constructor function is for the Model class. The class definition includes the classifier. The skeleton code in model.py is commented with the sections that you need to complete, or that you should review so that you understand what the code is doing.

def sentence2vec(self,sentence,dictionary):
This function is the one requiring the most work. You need to write a function to convert each review consisting of an array of tokens into a sentence representation vector. It should have two options, one for BOW and another for GloVe.

def load_glove(self):
This function loads the pre-trained GloVe vectors from the location where you unzipped the GloVe download from the Stanford page (see above).

def training(self):
This function splits the data so that some is reserved for testing, then trains and tests the model. It uses a cross entropy loss function and an Adam optimizer from the PyTorch module, as well as the components of a linear model.

You are encouraged to define other helper functions to modularize your code.

5 Step-by-Step Instructions

5.1 Download zip file

The zip file for this assignment is available from the Canvas assignment page. It includes the python project files and the dataset.

5.2 Project Tasks

Task 0: Create a ***python 3*** environment (e.g., in Anaconda, PyCharm, or your favorite IDE). Install all the dependent packages: ***torch, nltk, numpy***. You will also need to install the nltk data per these instructions. Specifically, it requires nltk_data/corpora/stopwords, nltk_data/tokenizers/punkt/PY3/english.pickle Make sure your code can run in this virtual environment.

Task 1: Write the functions to create ***GloVe*** and **B*OW*** word representations. Load the GloVe pre-trained word embeddings. Again, you can experiment with any of the GloVe embeddings from glove.6b.zip.

Task 2: Extend the classifier function and carry out experiments with BOW vectors and

GloVe-based text vectors. The existing code divides the reviews into 10% for testing and the rest for training, rather than using a held-out test set, so we can avoid Out-Of-Vocabulary (OOV) words in the test set (words that did not appear in training). You can experiment with command-line hyper-parameters of the classifier, such as the ***learning rate, hidden_size, embed_size.***

The code uses the python argparse module to define command line arguments (parameters). To run a BOW experiment after you have completed your model.py, you can use this command, where H is the size of your hidden layer (see the model.py code):

% python run.py --algo BOW --lr 0.001--hidden_size <H> --review_file ./data/reviews.txt --label_file ./data/labels.txt

To run a GloVe experiment, you need to have created the glove directory to store the glove.6B.<D>d.txt files, after which you can use this command, where H is the size of your hidden layer (see the model.py code), and D is the size of the GloVe embeddings you for this run:

% python run.py --algo GLOVE --lr 0.001 --embed_size <D> --hidden_size <H> --emb_file ./glove/glove.6B.<D>.txt --review_file ./data/reviews.txt --label_file ./data/labels.txt

Every training experiment may take from 5 to 30 mins to train and test each model depending on your machine, and on whether you increase the number of training epochs above the hard-coded value of 10. You do not need to wait for every training process to end automatically, that is, if the loss and accuracy have converged, you can stop the training.

Task 3: Answer the questions in **section 7**, using evidence from your experiments.

6 Testing

We will run your code submission to verify the results you submit. Do not import new modules that we would need to install; you can import any standard python module, e.g., **collections, itertools, math, random, queue, email, os, re, string, copy, sys**. If your code does not run in our environment, you are at risk of getting a zero grade.

Points for BOW: 30 if Acc > 90%

Points for GloVe: 30 if Acc > 80%

7 Report and Questions

Write a brief report that consists of a short introductory paragraph explaining your implementation, followed by a paragraph for each of the three numbered questions below.

The introductory paragraph should explain for your best results the BOW representation (e.g., what type of count), the GloVe representation, and the model or models. For the model or models, did you use the existing framework of 3 layers? What activation function did you select for each model? What loss function, and what optimizer? (**5 points**)

For the answers to the three multipart questions below, the length is not as important as providing a clear, and complete answer. Do not write too little to be complete, or so much that your answer lacks clarity. In the answers to questions 2 and 3, you can refer to any additional experiments you did that could not be included in the csv file of results (see section 8 Deliverables).

1. **(10 Points)** For each of the 2 to six rows in your csv file with experimental results (see section 8 Deliverables), provide a few sentences explaining what you varied in this experiment, and for each element you varied, what your motivation was. The things you vary should be intelligently chosen, thus you can vary the learning rate and the hidden state size without changing the basic code, if you think these should have an impact on the results that you either do see in your results, or for some reason do not see. You can vary the way the BOW representation is created, or the GloVe representation. You can vary the design of the learning

algorithm.

2. Why did the BOW representation achieve better accuracy on this problem than the GloVe representation **(10 Points)**? If you tried variant BOW representations, such as a weighted count rather than a raw count, compare the performance of the BOW representations (**2.5 points**).

3. Why did the GloVe representation achieve lower accuracy on this problem than the BOW representation **(10 Points)**? What was the most obvious advantage of GloVe over BOW? Which command line parameters had the most impact on performance, and why. If you tried modifications to the classifier, what did you try and what was the impact (**2.5 points**).

8 Deliverables

Submit a single zip file called <PSUID>.zip where PSUID is your psu email id (drop the "@psu.edu" portion.) The zip file should contain

- Your completed model.py file and any other supporting .py files, with no environment dependencies, and no subfolders. We will run your code locally to verify the accuracy results you report. If we cannot run your code, your grade might be zero.
- A csv file called ***<PSUID>.csv*** file with at least 2 lines for your runs and results, one for your best BOW run and one for your best GLOVE run. First put your command line for a run, then after the field-separator ',', put the accuracy your model achieved. We will run your code and expect the same accuracy, within some epsilon error. You can have as many as 6 runs total in this file, if you want to refer to your non-optimal results in the answers you provide to the questions in section 7 of the assignment.
- A pdf file named ***<PSUID>.pdf*** providing answers to the questions from section 7.