Assignment #3: Word Embeddings

Total score: 100
This assignment can be completed individually or by a team of a maximum of three members.
Due date: See the course page

Objectives

Build sentiment analysis models using Word2vec word embeddings and compare their classification performance with other vectorization methods.

Datasets

For this assignment, use the dataset aggregated_news.csv*(date, symbol, news)* created in Section 1.4 (b) of Assignment #2.

Organization of Program Files and Directories

The required program files and directory structure are provided in "Assignment_3.zip" for efficient grading. Before getting started, review the helper functions and test file. These files are provided to help you standardize and validate your CSV files. Each file in the directories is currently blank and must be completed by you. The file names correspond to the required tasks described in the sections below. **Do not change** any file names, directory names, or the directory structure. For example, the program "main.py" should be executable with the command: "**python** main.py" without requiring additional parameters or instructions.

Required Tasks

1 Vectorization of the News Data using Word Embedding

Complete the program "**embedding.py**" for the following task.

1.1 Skip-gram embedding using the genism package

(a) Create a Skip-gram-based word embedding from the news in "aggregated_news.csv" using the "Word2Vec" library available in the "gensim" package. Describe the key parameters used, including vector dimension (vector_size), window size (window), and minimum frequency (min_count).

(b) Vectorize the aggregated news using the trained word embedding and create a dataset with the following schema:

vectorized_news_skip-gram_embeddings*(date, symbol, news_vector, impact_score)*

- A small example code "word2vec.py" for Word2Vec embedding and classification are provided to help you understand the modeling process.
- To create a single news vector at each date from an aggregated news (headline + full news article), you can use a simple average of all word embedding vectors or TF-IDF.

1.2 ***Implementing CBOW embedding from scratch*** *(**optional** for bonus 20 points)*

(a) Implement the following gradient update rules for CBOW embeddings:

Hidden vector (for context representation):

$h = \frac{1}{C}\sum_i^C v_{w_i}$ where context size C, input embeddings v_{w_i} (for context words i)

Predicted probability for the target word w_t:

$\hat{y}_t = P(w_t|context) = softmax_t(h)$ applied over all vocabulary V)

Objective function (maximize likelihood of target word):

$\mathcal{J} = \log P(w_t|context) = \log \hat{y}_t$

Loss function (minimize negative log-likelihood):

$\mathcal{L} = -log\hat{y}_t$

Gradient w.r.t output embeddings: $\frac{\partial \mathcal{L}}{\partial v'_{w_j}} = (\hat{y}_j - y_j)h$ ($y_j = 1\ if\ j = t$, otherwise 0)

Update rule: $v'_{w_j} \leftarrow v'_{w_j} - \eta(\hat{y}_j - y_j)h$

Gradient w.r.t hidden/context vector: $\frac{\partial \mathcal{L}}{\partial h} = \sum_j^V (\hat{y}_j - y_j)v'_{w_j}$ (for all vocabulary j)

This is just the weighted sum of output embeddings, weighted by the *softmax* error.

Gradient w.r.t input embeddings: $\frac{\partial \mathcal{L}}{\partial v_{w_i}} = \frac{1}{C}\frac{\partial \mathcal{L}}{\partial h}$ (note: $\frac{\partial \mathcal{L}}{\partial h}$ is the gradient w.r.t hidden vector)

Update rule: $v_{w_i} \leftarrow v_{w_i} - \eta\frac{1}{C}\sum_j^V (\hat{y}_j - y_j)v'_{w_j}$

Note: implementing the hierarchical *softmax* and negative sampling are optional.

(b) Create a CBOW-based Word2Vec embedding without using existing libraries to create a word embedding from "aggregated_news.csv" and create a dataset: ***vectorized_news_cbow_embeddings****(date, symbol, news_vector, impact_score)*

2 Text Classification for Sentiment Analysis

2.1 MLP design and training

(a) MLP design

Complete the program "**model.py**" for the following task.

Design a MLP in Pytorch to train a classifier using the embeddings. Clearly describe the key architectural choices, including:

- Number of layers, Number of neurons per layer
- Activation functions
- Other hyperparameters (e.g., learning rate, batch size, optimizer, number of epochs)

(b) Training

Complete the program "**train.py**" for the following task.

Train the model using each of the **two embeddings** created in the previous section and save the trained models as "skipgram.pth" (and "cbow.pth").

2.2 Evaluation and discussion

(a) Complete the "**main.py**" so that your work is runnable with "python main.py". The "main.py" should:

- Generate datasets (for both Skip-gram and CBOW).
- Use the datasets to train your model and save the checkpoints.
- Evaluate the model.
- Print metrics

(b) In the report file, compare the results of your models with the classifiers built in "Assignment #2" and discuss the results:

- Compare the performance of the classifiers created by each of the embeddings in terms of accuracy (%). Briefly discuss whether the results are comparable and explain any performance differences.
- Does Word2Vec word embeddings improve the classification performance compared to the classifier built using the Document-Term Matrix (DTM)?
- Briefly explain the reason for improvement or lack of improvement in performance with embeddings.

Deliverables

- **One report file** (PDF or Word) that includes:
  - All team member names and % contribution of each member. If every member contributed equally, simply state "equal contribution." If there is disagreement on contributions, provide a brief task description for each member. In this case, different grades may be assigned individually.
  - All answers and results from the required tasks
  - When including example data in your report, show only a few representative samples instead of the entire dataset.
- **All Python program files** you created, **CSV files generated**, and **your .pth files**. DO not upload the original news datasets or Docker images. Please contact Tenshi (the grader) if the total size exceeds 1 GB. In that case, include only a portion of your datasets.

Grading Criteria

- Quality of work shown on the report (e.g., modeling process and results, methods used, and correctness of implementation and analysis)
- Level of understanding of related subjects as reflected in the report
- Effort (10%)