CS 590 NLP

HW3

Word2Vec, MLPs

Due 11/28 11:59 pm

Overall Goal:

In this homework you will work first with word2vec embeddings to understand them and then incorporate them with simple neural networks for classification. To save time, you will be again working with the datasets from HW2.

Useful python packages:
For word2vec, you may find the gensim package useful: https://radimrehurek.com/gensim/models/word2vec.html Note that gensim has built in pretrained word2vec embeddings so you should choose 1 or experiment with multiple in this homework. Gensim also has nice built in functions for working with word2vec data.
For the Multi-Layer Perceptron, there is a sklearn package: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html You may also implement this manually, but this would require also implementing the loss algorithm and stochastic gradient descent for training, so I would recommend against this.

Dataset
The dataset that will be used is the OLID dataset (same as HW2). This consists of offensive tweets, so be cautioned when viewing the dataset. You can find the training and test sets here: https://www.kaggle.com/datasets/feyzazkefe/olid-dataset Note for the word2vec and MLP tasks you will only need to worry about the first label (Offensive or Non-offensive) (Task A). The training set (**olid-training-v1.0.tsv)** will be used for training the MLP, while the test set (**testset-levela.tsv**) and labels **(labels-levela.csv)** will be used for testing and analyzing your models. Take time to understand how the training set and test set are laid out.

Final Report

Just like in the previous homework, you will be documenting observations and results in a final report. This is the best way to show your thought process throughout the homework, so it is best to update it as you go along and then refine at the end, rather than try and write all at the end.

Word2Vec Tasks

Your aim is to work with embeddings separately first to understand them a bit further. You may use any pretrained word2vec embeddings you choose.

compare_texts_word2vec(file_one, file_two, k = 10)
This method takes in two dataset files and compares them at the word level by leveraging word2vec. First, this function should find the k most common non stop words for each file. Then these 2*k words will be used to calculate 2 things:

1) Similarity between the 2 text files based on the top k words. It is up to you to determine how you want to calculate this similarity score, but cosine simlarity should be involved and the result should be a numerical score. (You should also document what algorithm you use).
2) Unique words for file_one, file_two and overlapping words for the two files. Rather than limiting to the 2*k words, you should additionally obtain the 10 most similar words for each k word leveraging word2vec to do so. This <= 10 *k words will then be used to find the required lists of words.

You should then print out a nice summary for the above statistics, no need to return anything. You may also use helper functions to help your function look nicer.

With the above function, you should then do the following function calls:

1. compare_texts_word2vec(NOT_subset, OFF_subset, k = 5)
2. compare_texts_word2vec(NOT_subset, OFF_subset, k = 10)
3. compare_texts_word2vec(NOT_subset, OFF_subset, k = 20)

The above statistics should all go into your final report document. Additionally, you should make some observations about both the similarity statistics and the words. Is anything surprising or not? (This should be more than just pointing out what the values are.)

MLP Tasks

These tasks are similar to the LR or NB tasks from HW2, however, you are now working with word2vec embeddings as the input and not TF-IDF.

You will create 2 required functions for MLPs (you may create more functions for your own use, but you need to at least create these two as specified):

train_MLP_model(path_to_train_file, num_layers = 2)

This method trains a multi-layer perceptron model on some training data and returns that trained model. The training texts should be represented by word2vec embeddings. You may use any pretrained word2vec embeddings you choose. Recall that the input size will affect how much of the input text is able to be sent in to the model. The MLP slides had some possible solutions, so you may choose any of these (but you should always note your decisions). **The format for the train file should follow the same format as the olid-training data file!**

test_MLP_model(path_to_test_file, MLP_model)

This method tests a trained MLP model on some test file and outputs a test file in the same format as the input test file but with 2 columns added: 1) probability of that text being offensive, 2) class prediction (OFF, NOT). **The format for the input file should follow that of the olid test file.**

Once these functions are implemented use them to accomplish the following tasks.

1. Train a 2 layer MLP model on the entire train set.
2. Test the trained model on the test set and produce predictions for all test texts.
3. Repeat 1 and 2 for a 1 layer MLP and a 3 layer MLP. Compare and contrast the overall accuracies for the models.
4. Write the analysis and comparison observations in a report document to be handed in alongside code. Note that even if changing something causes the accuracy to drop, this should be reported along with your reasoning of the drop/improvement in accuracy. This report is how you can demonstrate that you are thinking through the problem and give you practice for your final project.

Additional Tips/Guidance

1. The accuracy you achieve will not determine your final grade for this homework. Rather your thoughtfulness in approaching and analyzing your models. This means that lack in analysis will receive lower scores as indicated in the grading scale.
2. Note that preprocessing the text is up to you. This assignment is not specifically evaluating preprocessing like the first assignment, but here you have a chance to practice any preprocessing for your final project and future assignments. (**You should be doing some preprocessing, or else you'll be purposely setting your ML model up to fail.**)
3. Get familiar with the gensim and sklearn libraries on your own on small sets of data. Just copying and pasting the code without fully understanding it will end up being detrimental as you may not be able to accurately give proper analysis and observations of results (especially for the MLP model where you are expected to make improvements/changes).
    a. Do not just copy and past the sklearn class code. You may use portions from the sklearn classes, but too much unused code from classes will result in lost points due to poorly written code. Experiment with the code and only import what you need.

4. DO NOT SHARE CODE. I have linked sklearn libraries which will be useful for this assignment. You may also come to me to discuss/figure things out.
5. Start the assignment early. You will have 3 weeks to complete this assignment, which is plenty of time if you start early any familiarize yourself with the libraries. **IF YOU WAIT UNTIL THE LAST MINUTE TO START, YOU WILL BE LESS LIKELY TO DO WELL.** I won't be granting any additional extensions so each late assignment with follow the late grade policy.
6. Ask questions/approach like a researcher. Think like this is your chance to explore NLP models and analyze their effectiveness.

Grading

Assignment will be graded as follows:

Description	**Points**
Code Runs	10
word2vec Implementation/Tasks	25
MLP Implementation/Tasks	25
Report analysis and observations	25
Documentation (Comments, functions, etc)	15
Total:	**100**
Extra Credit (more word2vec tasks)	**20**

EXTRA CREDIT (20 pts): Improving Pretrained word2vec embeddings

For extra credit, you may fine-tune one of the built in word2vec embeddings on the training text of the OLID data. Essentially, you will be aiming to make the word2vec embeddings better representatives of the OLID data. You will then be using these updated embeddings in your MLP model to see if there are any improvements.

NOTE: Extra credit will not be counted for any late submissions!

You will create 1 required functions for this task (you may create more functions for your own use, but you need to at least create these two as specified):

update_embeddings(path_to_train_file)

This method fine-tunes a pretrained word2vec embedding (your choice) on the training text and then saves the fine-tuned embeddings. The function should then return the path to the fine-tuned embeddings. **The format for the train file should follow the same format as the olid-training data file!**

<u>You should also modify your MLP function to take in an argument for the word2vec embeddings to use.</u>

Once these functions are implemented and modified use them to accomplish the following tasks.

1. Train one set of pretrained word2vec embeddings on the OLID training text.
2. Test one of your trained MLP models on the test set producing scores for both the original pretrained embeddings and the fine-tuned embeddings. (Note that you will need to train one MLP on the fine-tuned and one on the pretrained.)
3. Compare the overall accuracies of the pretrained embeddings MLP to the fine-tuned embeddings MLP. Are the accuracies better, worse, or the same? Is this surprising or not? What might be causing the results?
4. Write the analysis and comparisons to your report file. This report is how you can demonstrate that you are thinking through the problem and give you practice for your final project.
5. You should provide a link to your fine-tuned embeddings on your final report. Since the embeddings are most likely too large to hand in on brightspace, you should find some place to host them (onedrive, google drive, etc.)

Note, there should be some apparent differences between the embeddings themselves. A simple way to check is by looking at specific words and the most similar words to them in both spaces. If you can't find differences, then there is most likely a problem with your fine-tuning process.