Spring 2025

EDS 6397 – Natural Language Processing

Assignment 3 – Sentiment Analysis with basic neural networks

Use of artificial intelligence assistant such as ChatGPT in developing the code for this assignment is allowed. You might still choose not to use AI assistant. However, ***the report needs to be completely written by you****. Use of grammar correction tools is disallowed. University of Houston academic honesty policy (included in course syllabus on Teams) applies.*

Assignment Overview

The objective of this assignment is to familiarize you with text classification using simple neural networks. More specifically, you will use feedforward neural networks (FFN) as well as recurrent neural networks (RNN) for this assignment. You will use Word2Vec embeddings to represent words as dense vectors.

Assignment Description

You will work with 5,482 short statements (sentences) that are gathered from financial news for sentiment analysis. These sentences are news headlines about companies being traded in the stock market. The news sentiments are labeled as negative, neutral, and positive. The goal is to train a neural network model that can analyze the sentiment of every news headline about companies that are traded publicly.

A few reminders:

- Make sure you follow the instructions below very carefully.
- Make sure you cite any online resource or paper that you used that helped you develop the code and/or report for this assignment.
- You need to develop your code using Keras. No other framework (basic tensorflow, pytorch, etc.) is allowed.
- Any time you need to tokenize, you need to use the tokenizer in Keras.preprocessing.text. No other tokenizers allowed.
- You need to use pretrained word embeddings from Word2Vec. Make sure to remove all words that do not exist in Word2Vec vocabulary from your own vocabulary (see step 2).

Input Data

You are given a csv file (assignment3_data.csv) which has 5,482 records of data in two columns; Sentence and Sentiment. There are 3 categories of sentiments, negative, neutral, and positive. You should map them to 0, 1, and 2 respectively.

Data was downloaded from https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis

And can be cited as: *Malo, Pekka, et al. "Good debt or bad debt: Detecting semantic orientations in economic texts." Journal of the Association for Information Science and Technology 65.4 (2014): 782-796.*

Step 1: Pre-processing

Cleanup the data as described below:

(1) Handling numerical values: Word2Vec only has vectors for the 9 digits [1, 2, 3, 4, 5, 6, 7, 8, 9]. Our goal is to convert all numerical values to one of these so that we can represent them later using the known embedding vectors in Word2Vec. Write a function that goes through a given sentence, finds all numerical sequences (including decimals) and converts them to number characters in [1, 2, 3, 4, 5, 6, 7, 8, 9]. Then replace the original numerical character spans with these number characters. If you sort, it might make sense to Word2Vec. So, for example

"The company raised $5.6 M and another EUR2.5b in funding so profits increased by 15%."

Will be converted to

"The company raised $**2** M and another EUR**1**b in funding so profits increased by **3**%."

(2) Handling financial terms and linguistics: Next, write a function that converts characters [k, m, b, t] that follow numerical values and converts them to [thousand, million, billion, trillion]. At the same time, make sure to take care of numerical monetary unit representations. So, the same function converts

'USD7m', 'EUR1b', 'CAD 2T'

To

7 million USD, 1 billion EUR, 2 trillion CAD

Notice that sometimes there is a space in the string and sometimes there isn't. Your function needs to handle both. Also notice how the monetary units are moved to the end of the sequence.

(3) Remove all punctuations and special characters

(4) Remove extra spaces

Note that NO lemmatization, stemming, or lower-casing tokens should be done. Stop words should NOT be removed.

Step 2: Creating an embedding matrix

Note: use Keras's tokenizer whenever needed.

(1) The goal here is to reduce the size of the embedding matrix of Word2Vec because it has 3 million vectors in it. It is computationally expensive to load it as weights of the input layer in Keras models. We only need a small embedding matrix that has as many rows as the vocab size of the given dataset. We will

create our vocabulary (a python dict) and using it, we will retrieve word embeddings from Word2Vec embeddings. You need to

- Use gensim package. Specifically, use gsnsim.downloader to download 'word2vec-google-news-300' which is a pre-trained word2vec model.
- Create a vocabulary from all the input data records. Note that you should remove all the words that do not exist in Word2Vec vocabulary.
- The first token (corresponding to index 0) in your vocabulary should be </s> which is a sentence separator token.
- Create an embedding matrix by retrieving the vector associated to each token in the new vocabulary from Word2Vec embeddings and storing it in this new embedding matrix.
- This embedding matrix will be used with our neural network models.

(2) Convert input sentences to vectors of embedding indices. This will form your **X** (data):

Write a code that goes through your data and convert each sentence to a vector that has the indices of word embeddings from the embedding matrix. For example, the sentence

Apple stock is down today

Will be converted to

[0, 0, 0, 0, 0, 101, 29, 44, 239, 400]

Note that preceding zeros are result of zero padding. You need to zero pad to make sure all input sequences have the same length. The sequence length should be set such that it is 2 tokens larger than the sentence with maximum length.

(3) Convert your labels to one-hot vectors using Keras's to_categorical function. This will be your **y** (labels) in a categorical fashion (stacked one-hot vectors) which is how we perform multiclass classification.

Step 3: Neural networks:

The first step before you develop your ANN models is to check your labels to see if you have class imbalance or not. Plot the histogram of the labels.

If you detect any class imbalance, Use Class_weights parameter in Keras models in order to tackle the class imbalance problem and make sure to justify your choice of weights. If you don't detect any class imbalance, explain your reasons.

1- FFN Classifier with pre-trained embeddings:

- Using Keras, create a feedforward network with at least two hidden layers for the task of 3-class classification.
- Note that Keras takes care of one-hot sampling for you. You need to bring n the embedding layer as weights for the Embedding layer as

  Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_length, weights=[Embedding_matrix], trainable=**False**),

- Use an 80-20 split with random state set to 42 using scikit-learn.
- Set the max number of training epochs to 50 but activate early stopping.
- Adjust your hyper parameters to avoid overfitting and underfitting as mush as possible. (drop out, regularization, learning rate reduction)
- Set the number of nodes on each layer by some trial and error. Same for batch size.
- Make sure to print the model summary so we can check the input output shapes of each layer.
- Make sure to include the model summary in your report.
- Set the loss to categorical cross entropy and track the accuracy metric.
- Report precision, recall and f1-score per class.

2- FFN Classifier with fine-tuning pre-trained embeddings:

- Similar to above steps but set the allow the weight of the first layer to be trainable.
- Include model summary in your report and briefly explain.

3- RNN with FFN Classifier head and using pre-trained embeddings:

- Make sure you try stacked RNNs (2 to 4) to see which works better. Only include the best in your notebook and report.
- As before, decide on all other parameters and hyper parameter by trial and error.
- You are allowed to design your classification head with any architecture you prefer.
- Include model summary in your report and briefly explain.

4- complex RNN with FFN Classifier head using pre-trained embeddings:

- Experiment with bidirectional RNN. Create your arbitrary architecture here. Either a simple bidirectional RNN, or a few of them stacked, or combination with stacked RNNs.
- Make sure you use SimpleRNN in a bidirectional sense from Keras (not LSTM).
- You are allowed to design your classification head with any architecture you prefer.
- Include model summary in your report and briefly explain.

5- RNN using LSTMs with FFN Classifier head using pre-trained embeddings:

- Similar to case 3 but this time use stacks of LSTM instead of simple RNN. Create your arbitrary architecture here.
- Include model summary in your report and briefly explain.

6- RNN using Bi-LSTMs with FFN Classifier head using pre-trained embeddings:

- Similar to case 4 but this time use stacks of Bi-LSTM instead of simple bidirectional RNNs. Create your arbitrary architecture here.
- Include model summary in your report and briefly explain.

7- Your choice of model architecture (a combination of above cases) and your decision to use pre-trained or trainable embeddings:

- Try a few different ideas inspired by the accuracy of above models to create your suggested architecture.
- Include model summary in your report and briefly explain.

The report

Your report can be up to 8 pages. You can include training plots, per-class assessment tables, and model descriptions of each of the 7 models. It also has to include the following. Not including them explicitly will cost you points.

- Explain how you built the vocabulary from the input data and by making sure the words existed in Word2Vec vocabulary
- Mention the total number of tokens in your created vocabulary (this includes the sentence separator token.
- Compare the two FNNs in terms of performance. You should use the same architecture and layer network parameters for both. The only difference would be that in one the embeddings layer is frozen, and in the other it isn't. Explicitly mention if fine-tuning Word2Vec embeddings improved the accuracy or not. Explicitly mention why.
- Compare models 3 and 4 in terms of performance metrics. Comment on model complexities.
- Compare models 5 and 6 in terms of performance metrics. Comment on model complexities.
- Draw a conclusion on models 3, 4, 5, and 6. Which one performed better overall?
- Explain your designed model (model 7) and compare it to previous models.

Questions

1- Why did the sentence separator token had to be the first token in the vocabulary dict that you created?
2- Check to see if your vocabulary has repeated words with different cases (e.g. *The* and *the*). Why do we have them both? Find 4 more examples of this and report here.
3- Check Word2Vec embeddings. Are word embeddings (from Word2Vec) similar for these words or are they different? Why?
4- Briefly explain what class imbalance is and what problems it might pose. Did you have class imbalance in this assignment? If yes, how did you handle it?

What to submit:

1- Your Python Jupyter Notebook should include plenty of comments and explanatory cells to ensure that your code is easy to read and understand. The Notebook should also contain the results of each cell executed by you. It should be named as follows: **[Last_name]_HW3.ipynb**
2- A report (4-8 pages) **in PDF format** that briefly discusses what you did for this assignment as well as the results. Make sure you include everything listed under subsection *the report* above as well as answer to all questions above.

Late submissions will be penalized at a rate of 1 point per hour. We round up the time to the nearest hour.

Make sure you click on submit so your assignment is "Turned In". Uploading the files doesn't mean you turned the assignment in.

Due Date: March 26, 2025 by 11:59 PM (submit through Teams)

Appendix A - How to install tensorflow-gpu along with genism using conda

I have included a yml file in the assignment folder that I created from my environment today. It may or may not work for you. If it doesn't, then you might want to follow steps below to create your environment and install packages.

You can navigate to the following website to get the version of cudatoolkit, cudnn, python, and tensorflow that work best for you machine.

https://www.tensorflow.org/install/source_windows#tested_build_configurations

For my Windows machine, I am using the following steps to create a conda environment and install tensorflow-gpu.

Below instructions are from this blog post. I have slightly edited it.

```
conda create -n tf_gpu python=3.10
conda activate tf_gpu

conda install conda-forge::cudatoolkit=11.2.2
conda install conda-forge::cudnn=8.1.0.77

pip install --upgrade pip
pip install tensorflow-gpu==2.10.0

conda install -c conda-forge pandas genism scipy scikit-learn Jupyter spyder
```

If that last statement doesn't work, try them with pip install