P1: Sentiment Analysis

Part I DUE: Thu, Aug **27th** at 11:59 am
Part II DUE: Sun, Sep **6th** at 11:59 am
Programming language to use: Python3 or Java

In this assignment, you will learn how to use word vectors to classify sentences based on the sentiment they express. Specifically, this assignment emphasizes long sentences, which are often challenging for existing sentiment analysis tools.

The assignment has two parts.

Part I (due on 11:59 am ET 8/27/20) involves each student manually labeling some sentences that we will provide. Specifically, each student will receive two CSV files containing ~45 sentences each). The instructions for labeling are provided below.

Part II (due on 11:59 am ET 9/6/20) involves each student proposing a classification model for a dataset that will be shared with you by the TA. The dataset contains ground truth aggregated by the TA from the crowdsourced labels from Part I. You will train and test a classifier and report performance on this dataset. This part emphasizes applying various NLP techniques and using popular word-embeddings such as GloVe, word2vec, and doc2vec for text classification.

Please do not slack off in Part I, as your work in Part I will affect the quality of your (and others) work in Part II.

The Task: Classifying the sentiment of long sentences

Sentiment analysis, also called opinion mining, is the field of study that analyzes the sentiments about something expressed in the text. An opinion consists of two key components: a *target* and a *sentiment* on the target. Sentiment analysis is the automatic analysis of text and tracking of the predictive judgments. The task of sentiment analysis is to classify whether the sentiment of a text is positive, negative, or neutral.

It can be nontrivial to analyze the sentiment expressed in long sentences. For example, a long sentence may not express any sentiment even though it contains sentiment words. Moreover, a positive or negative sentiment word may have opposite orientations in different situations. Thus, you will need to classify the sentiment according to your understanding of the context, and not merely by looking for any keywords. Below are examples of long sentences with sentiment words, and without obvious sentiment words, but for which you can infer the sentiment from the context.

Example 1: *When she got to the door, she found she had forgotten the little golden key, and when she went back to the table for it, she found she could not possibly reach it: she could see it quite plainly through the glass, and she tried her best to climb up one of the legs of the table, but it was too slippery; and when she had tired herself out with trying, the poor little thing sat down and cried.*
This is a long sentence with many sentiment words (like best, golden, little, poor) but doesn't express much sentiment. A traditional approach based on a sentiment lexicon would fail to identify the overall neutral sentiment of this sentence.

Example 2: *Having gotten a bit of a bad rap for not being a big box office hit like Pacino's previous film, " Scent of a woman, " and not having as strong a performance as he did in that one (he had just won an Oscar), " Carlito's Way " was destined for underrated heaven*.
The overall sentiment is discernible to a human despite the lack of obvious sentiment words. However, traditional sentiment analysis would fail to recognize the true sentiment of this review due to its sarcastic tone.

Part I: Manual Labelling (20 points)

In Part I, you have to manually score each sentence based on the sentiment it expresses. As this assignment focuses on long sentences you may find instances that have more than one sentiment within a sentence (perhaps towards different entities, e.g., *The movie was great, acting was ok*). In such cases, try to gauge the sentiment of the sentence as a whole, like for the given example the overall sentiment would be more towards positive than negative as *great* is more positive than *Ok* is negative in this context.

Provide sentiment scores for the sentences (in the CSV file shared with you via email) on a Likert scale of 1-5, 1 being the most negative, and 5 being the most positive. You will be

presented with (~90) long sentences from the Cornell movie review dataset. Provide one label for each long sentence. You need to score each sentence based on its overall context instead of the specific sentiment expressing keywords. For instance, the following sentence in spite of containing sentiment keywords like *hideous kinky', 'peace'*, *'love'* and *'faith'* should be labelled as neutral, as the sentence is describing the plot of a movie and not expressing any opinion.

> *Hideous kinky is not a plot-driven movie, but it tells the tale of a hippie named Julia (Kate Winslet) who has brought her two young girls (Bella Riza and Carrie Mullan) to Morocco to find peace, love, and faith in the early '70s.*

Deliverables for Part I:

- You have to label all instances shared with you on two CSV files.

Part I is due on Aug 27th at noon.

Part II: Classification (80 points)

For Part II we will share a labeled dataset with labels from crowdsourced labels collected in Part I. We will hold out a part of the labeled dataset and will share it at the end (a day before submission) for you to report your model's performance on the test dataset. We recommend you use Python3 (over Java) simply for the ease of use and the availability of countless libraries for NLP and machine learning, you can also use Java if you are more comfortable and confident in it. **Note**: there are only 0, 1, 2 labels in the dataset.

Word Embeddings and Vectors

Text is high-dimensional, unstructured data, which makes it computationally ill-suited and inefficient for processing in raw form. Word embeddings map a set of words or phrases in vocabulary to a vector of numerical values that is well-suited and easier for a machine to perform computations on.

Moreover, word embeddings are a form of word representation that is not only efficient to process but also often brings forth elements of meaning that are comprehensible to humans. Word embeddings have revolutionized NLP in recent years and are prominent in modern practice.

The following are some easy-to-use pre-trained word embedding models/libraries that you may use.

Google News word2vec: A pre-trained model that includes word vectors for a vocabulary of 3 million words and phrases that are trained on roughly 100 billion words from a Google News dataset. The vector length is 300 features.

GloVe *(Global Vectors for word representation)*: an unsupervised learning algorithm that generates word embeddings by aggregating global word-word co-occurrence matrix from a corpus.

Deeplearning4j: A Java library that implements word2vec, doc2vec, and GloVe word embeddings.

SpaCy word2vec: Spacy is a free, open-source Python framework for NLP. Spacy comes with a pre-trained word2vec model with a vocabulary of more than a million words.

Train your own word2vec: You can also train your own word2vec model using existing libraries. The Gensim library in Python and deeplearning4j in java provides an easy way to implement word2vec of your own. Training your own word2vec model can provide more flexibility and customizability for your specific task. If so, choose your training corpus carefully, your training corpus should have a similar vocabulary and use of words as the corpus you will be applying it on. For instance, a word2vec model trained on English poems (or Shakespeare) might not work well on news articles.

Many other word embeddings and word vector representations available, such as ELMo, Flair, dependency-based word embeddings, etc, feel free to explore those as well.

Baseline

Select **TWO** word embeddings (vectors/encoders) model in your baseline model**.** As a baseline model, you can use a word-embedding of your choice (or Tf-IDF) to vectorize text and a classification approach to classify each sentence based on its overall sentiment.

You need to take the following steps to get a baseline model (you may add more steps if you wish):

- Tokenize each sentence in word tokens.
- Compute vectors for each word token in a sentence and average these word vectors to get a vector for the sentence.
- Train a classifier to classify each long sentence into **0, 1, 2**.

We have specified basic steps that would give you a baseline model. Whatever performance your baseline model achieved try improving on that in your proposed solution. We have provided some pointers to give you some starting ideas but don't feel restricted to them. Try using your understanding of these sentences from your part I (labelling task) to help you come up with ideas for the proposed solution.

Proposed Solution

Propose **ONE** solution that you think would work better than the baseline. Here are some ideas, and feel free to work on your proposed model with other methods:

- The baseline model simply averages the vector scores which can't hold the context of the original ones. Thus, the sequential and complex structures are not reflected in averaging. Consider a sentence embedding technique, such as Universal Sentence Encoder and Doc2Vec. In this method, you need to **create a vector for each sentence** instead of converting tokenized sentences into vectors. More examples you can reach here: Use-cases of Google's Universal Sentence Encoder in Production.

  Example Code:

```
import tensorflow as tf
import tensorflow_hub as hub

embed = hub.Module("https://tfhub.dev/google/universal-sentence-encoder-large/3")

tf.logging.set_verbosity(tf.logging.ERROR)
with tf.Session() as session:
    session.run([tf.global_variables_initializer(), tf.tables_initializer()])
    X_train = session.run(embed(X_train))
    X_test = session.run(embed(X_test))
```

- Try different text preprocessing and observe how it impacts the performance of your model. Some basic text preprocessing you can try are **lemmatization**, **stemming**, **removing stop words**, and/or **punctuations**. You could find details from the lecture slides or the book Jurafsky&Martin. And there are some modules you could apply, like nltk by importing PorterStemmer or LancasterStemmer, take a look at the examples if you are interested in, Stemming and Lemmatization.

Report Requirement

1. Report and compare the performance of the **baseline** with **TWO** word embedding techniques, as well as the performance of your **proposed solution**.
2. In your report, you need to include **screenshots** of the **metrics** for performance at least with the accuracy and F1 score. Pick and choose other metrics as you see fit.
3. Include **code snippets** and explain what the method is on each requirement above: 1) code of method to tokenize, 2) code of method to calculate vectors, 3) code of generating the model you applied to classify training and testing dataset.

Deliverables

Upload a single zip file (make sure not to include any unnecessary superfluous files that may inflate the file size beyond the submission locker's limit) to the submission locker. Due on Sep 6th, 11:59 am (noon). Include the following in the zip file:

- A report that:

    - Describes your baseline solution and proposed solution in detail; and
    - Compares the performances of the baseline with **TWO** word embedding techniques, as well as your proposed solution.

- All source code with all files **includes** the pre-trained word embedding models you used.
- A ReadMe file to explain **each step** of the compilation and execution of your program, make sure the code will be executed successfully, otherwise, there will be a 20% deduction. Make a note when it is necessary to comment/uncomment code when switching pre-trained models and others.

Part II is due on Sept 6th at noon.