P1: Word Vectors

Part I DUE: Thu, Sep **5th** at noon
Part II DUE: Sun, Sep **15th** at noon
Programming language to use: Python3 or Java

In this assignment, you will learn how to use word vectors to complete a task, which we frame as a classification problem. Specifically, this assignment emphasizes techniques and tools such as GloVe, word2vec, and text classification.

To solve this task, the students with jointly create a dataset of ground truth for the classification via manual labeling (**Part I**). The combined dataset will be published. You will train and test your classifiers and report the performance (**Part II**) with respect to that dataset. Please do not slack off in **Part I**, as your work in **Part I** will affect the quality of your (and other's) work in **Part II**.

The Task: Ordering Events based on Common Sense

Relations between events have long been studied as they are the foundation of event inference, story understanding, and storytelling. Most studies focus on the temporal orders or causal relations between pairs of events. Some research measures the causal relations between events by the frequencies of their temporal orders. In other words, this research adopts the idea that two events are likely to be causally related if they happen more frequently in one temporal order than in the reverse order.

Instead of merely applying the statistics, we define the *inferential relation* between events. Two events are *inferentially related* if their sequential order can be inferred by common sense, and this order is therefore defined as an *inferential order*. Consider the following two sentences:

- S_1: I bought an apple and then ate it.
- S_2: I ate an apple and then bought it.

Based on common sense, "buying the apple" happens before "eating the apple." It is possible that you eat an apple in the store or a restaurant and pay for it later, but common sense would tell you that S_1 is the more likely or more normal scenario.

Here, we consider an *event* as a textual phrase, an *event phrase*, in a sentence that describes a single action. Note that some other studies define an event as an abstract object that refers to an actual incident that has taken place. For example, given the sentence “The earthquake killed more than 8,000 people,” an event detector may detect two events, *an earthquake* and the *killing*. However, we only consider the textual representation of the action, i.e., the entire sentence, which refers only to the killing.

In this project, your job is to infer inferential orders for pairs of events. You will create a dataset of inferentially ordered event pairs, and adopt classification techniques to determine the inferential orders between them.

Part I: Manual Labeling (20%)

You will be presented with (300) pairs of events. Provide one label for each pair:

- 1 if e_1 happens first,
- 2 if e_2 happens first, and
- 0 if you simply cannot decide the order without further information.
- (-1 if, on rare occasions, you do not think one of the phrases is actually an event.)

For example (the root verbs have been lemmatized),

Event 1	Event 2	Label
e_1: I (walk) out of the door.	e_2: I (open) the door.	2
e_1: I (glance) down.	e_2: I (see) my wallet.	1
e_1: I (move) to the front row.	e_2: I (put) on my glasses.	0

You can assume that both events being considered are extracted from one coherent story.

Interestingly, there are more types of temporal orders between events than “before” and “after.” Consider the following two sentences:

- S_1: I paid for the apple to buy it.

- S_2: I bought the apple to pay for it.

The two events, “paying for the apple” and “buying the apple,” should finish simultaneously. However, common sense tells us that S_1 is OK but S_2 sounds weird. You should label the events accordingly (paying before buying).

Submission for Part I:

- You have to label all instances shared with you on google sheet.

Part I is due on Sep 5th at noon.

Part II: Classification (80%)

After Part I, you will receive a training set and a testing set of labeled event pairs, combining all students’ results from Part I. You need to finish this part using these two datasets. We recommend you use Python3 (over Java) simply for the ease of use and countless libraries for NLP and machine learning, however, feel free to use Java if you are comfortable and confident in it.

Word Embeddings and Vectors

Text is high dimensional, unstructured data that makes it computationally ill-suited and inefficient for machines in raw form. Word embeddings are a form of word representation that tries to bridge the human understanding of language to that of a machine. Word embeddings map a set of words or phrases in a vocabulary to a vector of numerical values that is easier for a machine to perform computations on.

Some easy-to-use word2vec libraries in Python and Java are as following.

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)*: It is an unsupervised learning algorithm that generates word embeddings by aggregating global word-word co-occurrence matrix from a corpus.

Deeplearning4j: A Java library that provides implementation for word2vec, doc2vec and Glove word embeddings.

SpaCy word2vec: Spacy is a free, open source python framework to support Natural Language Processing tasks. 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. Gensim library in python and deeplearning4j in java provide 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 similar vocabulary and use of words as is in the corpus you will be using it on. For instance, training a word2vec model on english poems (or Shakespeare) might not work well on news articles.

There are 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.

You have to report results for 2 word embeddings (vectors/encoders) of your choosing.

Baseline

As a baseline, you are required to use word embedding and classification techniques to solve this problem. You need to take the following steps:

1. Tokenize both event phrases;
2. Convert each token into vectors;
3. Calculate the word vectors for event 1 and event 2, respectively, as the average of the vectors of the words that occur in each expression.
4. Concatenate the two vectors to obtain one vector for the event pairs;
5. Train a classifier to classify each event pair into Class 1, 2, or 0.

Note that, if you have a datapoint that says event 1 happens before event 2 ($<e_1, e_2> \rightarrow$ Class 1), it is safe to say you also have a datapoint that says event 2 happens after event 1 ($<e_2, e_1>$ $\rightarrow$ Class 2). You are required to expand both the training set and the testing set according to this rule. Include the statistics of the new training and testing sets in your report.

It is perfectly fine if the baseline does not yield satisfying results. Ordering events is a hard problem, and may not be solved by a small dataset.

Proposed Solution

Propose ONE solution that you think may work better than the baseline. Here are some ideas:

- Average word vectors do not consider the sequential order of the tokens ("I bought an apple and then ate it" and "I ate an apple and then bought it" have the exact same vectors). Consider a sentence embedding technique, such as Universal Sentence Encoder and Doc2Vec.
- Consider classification techniques that can be applied to sequences, such as RNN (LSTM), CNN, etc.
- Our labeled dataset is limited in size. You can propose ways of leveraging existing language models that have been trained on much larger corpora, such as Google's BERT and Facebook's PyText, for the classification. Refer to this link.

Report and compare the performance of the **baseline with two word embedding techniques**, as well as the performance of your **proposed solution**.

One required metric for performance is the accuracy of the classification. Pick and choose other metrics as you see fit.

Submission for Part II

Upload a single zip file (make sure not to include any unnecessary superfluous files that may inflate the file size beyond submission locker's limit) to the submission locker. Due on Sep 15th, 12:00 pm (noon). Include the following in the zip file:

- A report that:
  - Describes your proposed solution in detail; and
  - Compare the performances of the baseline with TWO word embedding techniques, as well as your proposed solution.
- All source code.
- A ReadMe file to explain the compilation and execution of your program.

Part II is due on Sep 15th at noon.