CSC495/791 Natural Language Processing

Assignment P1

Introduction

User generated content on the internet, like blogs, product reviews, and user forums are a rich source of diverse opinions on a wide range of topics. Online user reviews on products often contain comparisons (with rival products) that provide a more fine-grained insight about user preferences.

Identifying user *preferences* and *comparisons* in product reviews involves two tasks.

Task 1: Comparative Sentence Identification (CSI) *(Jindal and Liu, 2006)*
Given a sentence determine whether it contains comparison between two entities, and

Task 2: Comparative Preference Classification (CPC) *(Ganapathibhotla and Liu, 2008)*
Identifying the preferred entity in a comparative sentence containing two entities being compared.

In this assignment you will study both these tasks on a new dataset of online user reviews for comparative software products (apps and web-services).

Earlier work on CSI and CPC *(Jindal and Liu, 2006)*, *(Ganapathibhotla and Liu, 2008), (anchenko et al. 2019) (Kessler et al. 2014)* contain explicit comparisons when both entities being compared are mentioned in the text. However, this may not always be true, as online customer reviews are often very short and do not explicitly mention both entities being compared (i.e. implicit comparison, assuming the entity being reviewed).
For example, consider the following consumer review,

Way better than Pandora.
(from a review on Spotify)

The consumer expresses a preference without getting into the details of the aspects of comparison, and without explicitly mentioning both entities in the text, we consider such comparisons as *implicit comparisons*. Our dataset differs from previous works in that it includes many implicit comparisons that previous proposed datasets are lacking.

Dataset

We curated a sentence level dataset from product reviews identifying sentences expressing a preferred entity or showing a comparison between two entities. For the purpose of this assignment stick with the definitions provided below for comparison and preferred entity.

Comparison sentence is defined as one containing similarity, dissimilarity or preference between two entities. A comparison can be implicit (mentioning only one entity being compared) or explicit (mentioning both entities being compared, including pronominal references like, it, them, this, etc). For example,

1. If u like Temple Run u will like this

   (From a review on Subway Surfers)
2. Great music, lots of selections, plays out-side of the app and a lot better then Spotify.

   (From a review on Amazon Music)

Example 1 shows an explicit comparison between the two entities (*this* for Subway Surfer), while example 2 shows an implicit comparison with only one entity (Spotify) mentioned in the text.

Preferred entity is defined as the entity preferred over the other either based on the comparison (implicit or explicit) or if a preference is explicitly revealed. We consider cases where preference cannot be determined (non-gradable comparatives used or no clear preference can be inferred as in example 1a above) the same as no preference.

Part 1 (20 points)

Part 1 of the assignment involves annotating a sample dataset (***shared with you through google file, check your ncsu email and google drive***). You are required to label a sample dataset containing X data instances (sentences to be annotated) based on these Annotation instructions.

Annotating a dataset is an essential part for any classification task and since natural languages allow us to express the same information in so many ways make this task all the more interesting and sometimes challenging. Do not slack on this task as this will impact part 2 of this assignment that makes use of the annotations you provide.

P1 is due on 5:00 pm(afternoon) ET 9/5/21

Part 2 (80 points)

For part 2 we will share a labelled dataset collected in part 1. We will hold out a part of the labelled dataset and share it at the end (a day before submission) for you to report your models performance on the test dataset. We recommend you use Python3 simply for the ease of use and the availability of countless libraries for NLP and machine learning. There are two subtasks in part 2 corresponding to the two labelling tasks in part 1. These two tasks are essentially the same from the model perspective.

- Task 1: <u>Comparative Sentence Identification (CSI): this is a 3-way classification(0/1/2)</u>
- Task 2: <u>Comparative Preference Classification (CPC): this is also a 3-way classification(O/T/N)</u>

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 a 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 are not only efficient to process but also often bring 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 matrices 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 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 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.

Report results for two word-embeddings (vectors/encoders) of your choosing.

Baseline (50 points)

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. (10 points)
- Compute vectors for each word token in a sentence and average these word vectors to get a vector for the sentence. (10 points)
- Train classifiers for the two tasks. (30 points)

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 1 to help you come up with ideas for the proposed solution.

Proposed Solution (30 points)
Propose ONE solution that you think would work better than the baseline. Here are some ideas:

- The baseline model simply averages the vector scores which can't hold the context of the original ones. Thus, the sequential and complex structure are not reflected with 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 each token into vectors. More examples you can reach here: Use-cases of Google's Universal Sentence Encoder in Production.

- 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 can try coming up with text features that can be used for this task? For instance, if you are given the task of identifying questions, a good text feature would be words like, *'what'*, *'why'*, *'how'* and punctuation '*?*' to classify whether a text is a question or a non-question. Do similar cues exist that can be used to identify sentiments (other than the sentiment lexicons) in long sentences?

- 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**. 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. **Sklearn metric confusion matrix and classification_report are good starting points**.

Resources

We highly recommend you take advantage of the free computing resources from google Colab. It is an online jupyter notebook engine with free GPU resources which should be sufficient for this assignment. Learn more from their official tutorial.

Deliverables for Part II

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. Include the following in the zip file:

- A report that:
    - Describe your baseline solution and proposed solution in detail. Organize your report into different sections such as data source (if you use extra data), preprocessing, tokenization, modeling building, modeling training, modeling evaluation, results and so on.

    - Compare the performances of the baseline with TWO word embedding techniques, as well as your proposed solution.

- A jupyter notebook that:
    - Self-contained. We will download your notebook and run in Colab. You can assume the data is in the current directory (no need to manipulate the paths). You need to test your code before you submit, and make sure the dependencies are properly installed at the beginning cell. E.g. !pip install Spacy(in Colab, ! means shell commands)
    - Well-documented but not too verbose. Organize your notebook into sections such as data processing, text tokenization, model building and so on. Also make comments with your code to explain what your code does. You should not expect the instructor to speculate on your code.
    - Show some key intermediate execution outcomes. When you save your Colab notebook, the outcome of each cell is also saved automatically. You can remove some trivial outcomes such as installation information as you see fit.
    - Take a look at the format and organization of this notebook.

P2 is due on 5:00 pm (afternoon) ET 9/19/21