Text Analytics: 2nd Assignment

Tsirmpas Dimitris
Drouzas Vasilis

February 9, 2024

Athens University of Economics and Business
MSc in Data Science

Contents

1 Introduction

This report will briefly discuss the theoretical background, implementation details and decisions taken for the construction of MLP models for sentiment analysis and POS tagging tasks.

This report and its associated code, analysis and results were conducted by the two authors. Specifically, the sentiment analysis task was performed by Drouzas Vasilis, and the POS-tagging task by Tsirmpas Dimitris. This report was written by both authors.

Note that due to the relative custom code complexity, most of the code used in this section was developed and imported from python source files located in the ‘tasks‘ module. In-depth documentation and implementation details can be found in these files.

2 POS Tagging

POS tagging is a language processing task where words in a given text are assigned specific grammatical categories, such as nouns, verbs, or adjectives. The objective is to analyze sentence structure.

In this section we describe how we can leverage pre-trained word embeddings to create a context-aware MLP classifier.

2.1 Dataset

Acquiring and preprocessing our data with the goal of eventually acquiring a sufficient representation of our text is the most difficult and time-consuming task. We thus split it in distinct phases:

- Original dataset acquisition and parsing
- Qualitative analysis and preprocessing
- Transformation necessary for the NLP task

Each of these distinct steps are individually analyzed below.

2.1.1 Acquisition

We select the English EWT-UD tree, which is the largest currently supported collection for POS tagging tasks for the English language.

This corpus contains 16622 sentences, 251492 tokens and 254820 syntactic words, as well as 926 types of words that contain both letters and punctuation, such as ’s, n’t, e-mail, Mr., ’s, etc). This is markedly a much higher occurrence than its siblings, and therefore may lead to a slightly more difficult task.

The dataset is made available in `conllu` format, which we parse using the recommended `conllu` python library. We create a dataframe for every word and its corresponding POS tag and link words belonging to the same sentences by a unique sentence ID. The data are already split to training, validation and test sets, thus our own sets correspond to the respective split files.

We are interested in the UPOS (Universal Part of Speech) tags for English words.

Set	Mean	Std	Min	25%	50%	75%	Max
Training	16.31	12.4	1	7	14	23	159
Validation	12.56	10.41	1	5	10	17	75
Test	12.08	10.6	1	4	9	17	81

Table 1: Summary and order statistics for the number of words in the sentences of each data split.

Set	Total Word Count	Total Sentence Count
Training	204614	12544
Validation	25152	2001
Test	25096	2077

Table 2: Total text volume of each data split.

2.1.2 Qualitative Analysis

Our training vocabulary is comprised of 16654 words. We include qualitative statistics on the sentences of our dataset in Tables 1 and 2. The splits are explicitly mentioned separately because the splitting was performed by the dataset authors and not by random sampling. We would therefore like to confirm at a glance whether their data are similar.

2.1.3 Preprocessing

Given the nature of our task we can not implement preprocessing steps such as removing punctuation marks, stopwords or augmenting the dataset. Thus, the only meaningful preprocessing at this stage would be converting the words to lowercase. We believe that the context of each word will carry enough information to distinguish its POS tag regardless of case.

Another issue we need to address before continuing is that of words being part of (depending on) other words for their POS tag. Those would be words such as ”don’t”, ”couldn’t” or ”you’re”. In the standard UPOS schema these are defined as two or more separate words, where the first is represented by its standard POS tag, and the rest as part of that tag (UPOS tag ”PART”). For instance, ”don’t” would be split into ”do” and ”n’t” with ”AUX” and ”PART” tags respectively. In our dataset, these words are represented both in the manner described above followed by the full word (”don’t”) tagged with the pseudo-tag ”_”. We remove the latter representation from the working dataset.

The general algorithm to calculate the window embeddings on our dataset can be found in Algorithm 1. The algorithm uses a few external functions which are not described here for the sake of brevity. `get_window()` returns the context of the word inside a sentence, including padding where needed, `embedding()` returns the word embedding for a single word and `concatenate` returns a single element from a list of elements. The rest of the functions should be self-explanatory. Note that this algorithm does not represent the actual python implementation.

Algorithm 1 Window Embedding creation algorithm from raw-text sentences.

Input sentences, window_lim: a list of sentences and an upper bound of windows to be computed

Output tuple(windows, targets): the window embeddings and the POS tag corresponding to the median word of each window

```
windows = list()
targets = list()

for sentence in sentences do
    for word in sentence do
        window = get_window(word, sentence)
        target = get_tag(word)
        windows.add(window)
        targets.add(target)
    end for
end for

window_embeddings = list()
for window in windows do
    if window_embeddings.size ≥ window_lim then
        break
    end if

    word_embeddings = list()
    for word in window do
        if word is PAD_TOKEN then
            word_embeddings.add(zeros(embedding_size))
        else
            word_embeddings.add(embedding(word))
        end if
    end for
    window_embedding = concatenate(word_embeddings)
    window_embeddings.add(window_embedding)
end for

targets_vec = list()
for target in targets do
    targets_vec.add(one_hot(target))
end for

return window_embeddings, targets_vec
```

2.2 Baseline Classifier

We create our own classifier which classifies each token by the majority label associated with it. The classifier is defined as a subclass of sklearn's classifier superclass and thus can seamlessly use it in most sklearn-provided functions such as `classification_report()` and its implementation can be found in the `tasks.models` module.

The results of the classifier can be found in Figure TODO. These results make intuitive sense, since most words in the English language can be classified in a single label, irrespective of context. For example, "is" will always be classified as "AUX", and all punctuation marks will be classified as "PUNCT".

Thus, besides quantitative statistics such as categorical accuracy and f1-score, we should pay close attention to the precision and recall statistics for the more variable POS tags such as "NOUN" or "VERB" in order to properly evaluate our MLP classifier.

2.3 MLP Classifier

2.3.1 Hyper-parameter tuning

We use the `keras_tuner` library to automatically perform random search over various hyper-parameters of our model.

The parameter search consists of:

- The depth of the model (the number of layers)
- The height of the model (the number of parameters by layer)
- The learning rate

The parameter search does NOT consist of:

- Dropout rate, since dropout rarely changes the final result of a neural network, but rather tunes the tradeoff between training time and overfit avoidance
- Activation functions, since they rarely significantly influence the model's performance

With this scheme we hope to maximize the area and granularity of our search to the hyper-parameters that are most likely to significantly influence the final results.

We implement early stopping and set a maximum iteration limit of 70. We assume that if a model needs to go over that limit, it may be computationally inefficient, and thus less desirable compared to a slightly worse, but much more efficient model.

2.3.2 Results

TODO