CENG463 Homework-2

Elif Ecem Ümütlü - 2448991

January 14, 2024

Abstract

This homework has 2 parts each explained below under Task1 and Task2 titles.

1 TASK 1

In my implementation, I focused on increasing the values of dep_uas, and dep_las which are correspondent of the dependency unlabeled attachment score, dependency labeled attachment score, respectively. Other columns in the table is not my main concern yet used for developing the model further.

I want to emphasize that, the results I shared are under experiment environment where transformers used. Without transformers (only CPU), I could only managed to get around 64.5 and 53 from dep_uas, and dep_las, respectively. Moreover, I used colab environment to train the model. As a result, gpu activation id is referred to the colab environment.

1.1 Steps

1.1.1

Firstly, I converted train, development, and test sets to .spacy format which is actually DocBin format for spacy to be used. While converting them, I grouped sentences where each group contains 10 sentences so that GPU's parallel processing feature can be used more efficiently.

1.1.2

Secondly, I initialized the configuration file. Configuration file holds the structure of the model in a usable format by spacy. What determines the heart of the model is pipeline part of the configuration file. Each pipeline component listens previous components. Additionally, all of them listens the transformer which initially creates the word embeddings. I preferred to use the below pipeline.

transformer → tagger → morphologizer → trainable lemmatizer → NER → parser

This way, parser can be fed with the results of the previous tasks, which will help to improve its accuracy. This pipeline provides some kind of a feature selection. Dependency parsing task gives better results when inferring arcs not only from parser, but also some additional features like the name entity of the word in the sentence.

1.1.3

Thirdly, I initialized labels beforehand so that the spacy won't have to preprocess the data to extract the labels while training. (It is suggested on the spacy documentation.)

1.1.4

Fourthly, I trained the model with the config file I generated. After some tests, based on my observations, I made hyperparameter tuning for the model with the below configurations:

training.max_epochs 300

nlp.batch_size 128

training.dropout 0.2

training.patience 1000000 : default was 1600 which also causes to training to stop early

training.eval_frequency 200

components.trainable_lemmatizer.min_tree_freq 1

gpu-id 0

1.2 Results

1.2.1 Training results

In the below figure, we are interested in the values under the column name "DEP_UAS" and "DEP_LAS". As can be depicted from the figure, the maximum UAS, and LAS score development set get from the trained model is 75.48 and 66.77, respectively.

```
ℹ Saving to output directory: output_transformer_epoch300
ℹ Using GPU: 0

=========================== Initializing pipeline ===========================
✔ Initialized pipeline

============================= Training pipeline =============================
ℹ Pipeline: ['transformer', 'tagger', 'morphologizer',
'trainable_lemmatizer', 'ner', 'parser']
ℹ Initial learn rate: 0.0
E    #       LOSS TRANS...  LOSS TAGGER  LOSS MORPH...  LOSS TRAIN...  LOSS NER  LOSS PARSER  TAG_ACC  POS_ACC  MORPH_ACC  LEMMA_ACC  ENTS_F  ENTS_P  ENTS_R  DEP_UAS  DEP_LAS  SENTS_F  SCORE
---  ------  -------------  -----------  -------------  -------------  --------  -----------  -------  -------  ---------  ---------  ------  ------  ------  -------  -------  -------  ------
  0       0        2478.06      1477.59        1497.66        1498.55      0.00      3482.20     3.14     9.70      36.63      49.18    0.00    0.00    0.00    14.14     7.01     0.20    0.17
  9     200      350565.02    324347.16      348411.37      350091.70      0.00    365412.35    54.41    44.61      33.06      49.08    0.00    0.00    0.00    56.71    39.09    81.19    0.38
 19     400      203988.79    128969.23      240944.90      244758.68      0.00    191888.51    90.31    92.11      76.00      58.77    0.00    0.00    0.00    69.62    59.52    89.63    0.60
 29     600      128048.47     23723.68       81529.91      147995.86      0.00    123692.53    94.29    95.22      88.46      70.45    0.00    0.00    0.00    72.77    63.74    94.05    0.65
 38     800       83046.74      7019.08       37777.15      108507.96      0.00     83773.73    94.98    95.86      91.29      77.75    0.00    0.00    0.00    73.17    63.85    94.41    0.67
 48    1000       48587.74      2682.96       21720.19       74658.04      0.00     54029.75    95.41    96.25      92.36      81.64    0.00    0.00    0.00    73.47    64.68    95.67    0.68
 58    1200       31871.03      1259.80       13115.41       49776.37      0.00     40116.82    95.19    96.23      92.62      83.97    0.00    0.00    0.00    72.98    64.42    94.01    0.68
 68    1400       18996.06       618.46        7206.55       31155.19      0.00     31968.02    95.09    96.27      92.74      85.87    0.00    0.00    0.00    73.20    64.37    95.60    0.69
 78    1600       13945.35       319.18        3316.06       17694.24      0.00     29913.03    95.42    96.32      93.18      86.65    0.00    0.00    0.00    73.76    65.02    94.06    0.69
 87    1800       11745.14       175.57        1350.66        8732.85      0.00     28646.11    95.32    96.22      92.91      86.97    0.00    0.00    0.00    73.96    65.22    94.53    0.69
 97    2000        8077.43       109.97         534.72        3491.22      0.00     26398.15    95.25    96.23      92.95      87.15    0.00    0.00    0.00    74.15    65.40    95.37    0.69
107    2200        6659.32        69.33         224.17        1183.57      0.00     26376.10    95.32    96.25      93.06      87.40    0.00    0.00    0.00    74.46    65.85    96.42    0.70
116    2400        7107.28        67.18         127.98         396.87      0.00     26562.40    95.42    96.26      93.05      87.47    0.00    0.00    0.00    74.23    65.57    95.77    0.69
126    2600        5424.73        33.86          65.98         146.36      0.00     26013.05    95.45    96.33      93.14      87.43    0.00    0.00    0.00    74.01    65.47    95.99    0.69
136    2800        5304.44        39.75          44.85          55.21      0.00     25919.48    95.42    96.28      93.09      87.47    0.00    0.00    0.00    74.38    65.87    95.62    0.70
146    3000        4068.39        16.47          39.26          28.95      0.00     25379.23    95.30    96.28      93.06      87.46    0.00    0.00    0.00    74.45    65.71    95.22    0.70
156    3200        4184.85        13.95          21.99          19.14      0.00     25754.55    95.31    96.42      93.09      87.51    0.00    0.00    0.00    74.26    65.56    94.83    0.69
165    3400        4343.00        23.75          20.47           8.78      0.00     26014.35    95.50    96.41      93.12      87.48    0.00    0.00    0.00    74.75    66.04    95.24    0.70
175    3600        3700.56        11.53          15.35          10.22      0.00     25304.69    95.55    96.34      93.08      87.72    0.00    0.00    0.00    74.44    65.69    96.78    0.70
185    3800        2930.45        20.09          10.83           3.05      0.00     24864.38    95.55    96.34      93.10      87.55    0.00    0.00    0.00    74.58    65.86    95.74    0.70
195    4000        2988.42        14.28          14.22           6.63      0.00     25314.35    95.52    96.30      93.14      87.39    0.00    0.00    0.00    74.57    66.08    95.29    0.70
204    4200        2858.41         7.98          10.21           5.58      0.00     25219.84    95.44    96.28      92.91      87.39    0.00    0.00    0.00    74.87    66.26    96.18    0.70
214    4400        2195.39         4.80          11.11           5.03      0.00     25267.57    95.59    96.46      93.03      87.46    0.00    0.00    0.00    75.48    66.77    97.08    0.70
224    4600        2809.14         7.99          10.75           7.75      0.00     24778.99    95.56    96.31      93.08      87.41    0.00    0.00    0.00    74.92    66.29    95.60    0.70
234    4800        2432.59        26.88          29.17           6.98      0.00     25223.25    95.68    96.45      93.02      87.30    0.00    0.00    0.00    74.66    66.08    95.74    0.70
243    5000        2087.43         9.61          13.79          11.05      0.00     25445.38    95.50    96.33      92.93      87.46    0.00    0.00    0.00    74.83    66.11    96.12    0.70
253    5200        2148.92        14.74          14.42           5.96      0.00     25234.76    95.64    96.49      93.11      87.41    0.00    0.00    0.00    74.57    65.88    96.73    0.70
263    5400        1508.09         3.30           4.65           3.83      0.00     25057.12    95.57    96.40      93.16      87.40    0.00    0.00    0.00    74.90    66.40    96.14    0.70
273    5600        1775.12         2.31           7.08           4.78      0.00     25287.85    95.66    96.37      93.06      87.41    0.00    0.00    0.00    75.36    66.73    97.13    0.70
282    5800        1477.73         2.24           5.47           2.97      0.00     25427.57    95.63    96.35      93.08      87.43    0.00    0.00    0.00    75.29    66.55    95.94    0.70
292    6000        1451.40         4.18           6.85           2.31      0.00     24964.82    95.69    96.43      93.04      87.50    0.00    0.00    0.00    75.08    66.49    95.58    0.70
✔ Saved pipeline to output directory
output_transformer_epoch300/model-last
```

1.2.2 Test set evaluation results

When I evaluate the test set on the trained model, I get the below results with "UAS" 77.42 and "LAS" 68.73.

```
ℹ Using GPU: 0

================================== Results ==================================

TOK      99.82
TAG      95.56
POS      96.33
MORPH    93.01
LEMMA    87.53
UAS      77.42
LAS      68.73
NER P    -
NER R    -
NER F    -
SENT P   95.27
SENT R   96.93
SENT F   96.09
SPEED    2065
```

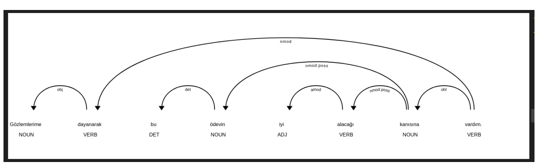
UAS and LAS have remarkable differences. UAS and LAS differ from each other due to one main reason. We are training a model since the dependency arcs cannot be figured out in a systematic way. A model for dependency parsing is also predicts the **relation (label)** between two words whereas it locates them under a node. Hence, the predicted label (relation between words) may not be correct. Nevertheless, the model can predict more easily whether there is a relation between buffered words or not (dependency). In the calculation of LAS, the correctness of the label is also included. Hence, LAS score is lower than the UAS. Here is how UAS and LAS are calculated:

$$UAS = \frac{Number\ of\ correctly\ calculated\ dependencies}{Total\ number\ of\ words}$$

$$LAS = \frac{Number\ of\ correctly\ calculated\ dependencies\ with\ correct\ labels}{Total\ number\ of\ words}$$

1.2.3 Testing on different sentences

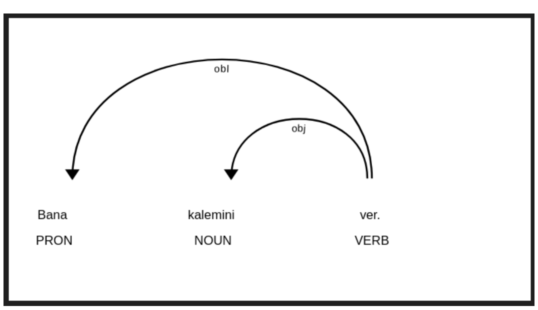
Here are some cases I tried on dependency parsing using my model. For the example images, first of them is calculated using the openlibrary *tr_core_news_lg* dependency parser model. The second image represents the results of my model. In most cases, my parser correctly found the dependencies whereas there are some problems predicting the relation type, as expected by looking at the UAS and LAS scores. Those falsely predictions may occur due to the lack of similar train cases in the dataset.



(a) Other model

(b) My parser

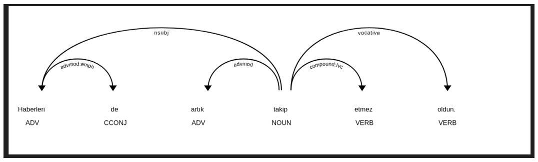
Figure 1: Mostly correct parsing



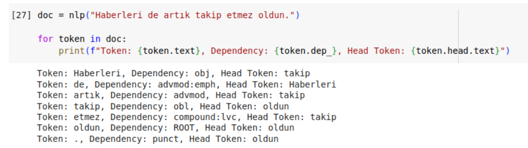
(a) Other model

(b) My parser

Figure 2: Correct parsing



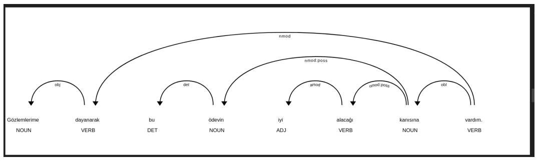
(a) Other model



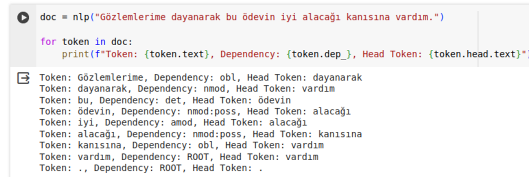
```
[27] doc = nlp("Haberleri de artık takip etmez oldun.")

     for token in doc:
         print(f"Token: {token.text}, Dependency: {token.dep_}, Head Token: {token.head.text}")

     Token: Haberleri, Dependency: obj, Head Token: takip
     Token: de, Dependency: advmod:emph, Head Token: Haberleri
     Token: artık, Dependency: advmod, Head Token: takip
     Token: takip, Dependency: obl, Head Token: oldun
     Token: etmez, Dependency: compound:lvc, Head Token: takip
     Token: oldun, Dependency: ROOT, Head Token: oldun
     Token: ., Dependency: punct, Head Token: oldun
```

(b) My parser

Figure 3: Partially incorrect parsing



(a) Other model



```
doc = nlp("Gözlemlerime dayanarak bu ödevin iyi alacağı kanısına vardım.")

for token in doc:
    print(f"Token: {token.text}, Dependency: {token.dep_}, Head Token: {token.head.text}")

Token: Gözlemlerime, Dependency: obl, Head Token: dayanarak
Token: dayanarak, Dependency: nmod, Head Token: vardım
Token: bu, Dependency: det, Head Token: ödevin
Token: ödevin, Dependency: nmod:poss, Head Token: alacağı
Token: iyi, Dependency: amod, Head Token: alacağı
Token: alacağı, Dependency: nmod:poss, Head Token: kanısına
Token: kanısına, Dependency: obl, Head Token: vardım
Token: vardım, Dependency: ROOT, Head Token: vardım
Token: ., Dependency: ROOT, Head Token: .
```

(b) My parser

Figure 4: Partially incorrect parsing

2 TASK 2

2.1 Implementation details

I used TFIDFVectorizer of sklearn library. I collected documents under a megadoc list where each element in the list is a string made out of all the sentences of a document. The base The TF-IDF vectorized matrix is obtained by transforming the megadoc using the vectorizer. Afterwards, I transform the sentences of the given document according to the vector calculated by megadoc. Afterwards, I use this vectorized sentence structure of a document to calculate the cosine similarity as follows: cosineSim(vectorized document sentences, vectorized document sentences). Following this, cosine similarity scores of the sentences are sorted and 5 most informative ones are selected among them.

In the TF-IDF vectorizer, I used two features which are stop words and max df. Former one is used for eliminating the stop words in English, and the latter one is used for ignoring terms that appear in the 80 percent of the documents.

I don't think so that algorithm works pretty good on documents with larger sentence count. The summarization model has difficulty to find the most informative sentences when there are too much informative sentences as in the case of large sentence counted documents.

The word selection of the algorithm is informative, but In my summarizations, I focused on different sentences other than algorithm picked. However, the algorithm cannot be directly evaluated as uninformative.

In my sentence selection, I picked longer sentences that includes human names (especially the name of the defendant). Since, the model simply just looks at the cosine similarty, it is not expected to get accurate results from it.

2.2 Experiments

Here are the files I used to check my extractive summarization code.

06_4.xml

My sentences id="s120" id="s195" id="s314" id="s365" id="s426"
06_11.xml

My sentences id=”s47” id=”s58” id=”s69” id=”s148” id=”s190”
09_801.xml

My Sentences id=”s18” id=”s46” id=”s77” id=”s102” id=”s108”
09_1505.xml My sentences id=”s28”¿ id=”s34”¿ id=”s38”¿ id=”s44”¿ id=”s47”¿
09_1502.xml My Sentences id=”s10”¿ id=”s12”¿ id=”s19”¿ id=”s20”¿ id=”s47”¿

2.3 Enhancements

Summarization task is mostly based on finding sentence scores out of its document. As I have searched through the papers, there are loads of different approaches to this task. Basically, other methods focus on sentence rescoring, and word revectorizing by either statistical approaches or machine learning approaches. Additionally, some methods use graph-based algorithms for sentence scoring.

I think, while doing extractive summarization, we should not be solely relying on the cosine similarity. Even though it informs us, this is not enough to obtain the most informative sentences. Not only the sentence similarity between sentences of document should be considered but also the location of the sentences, the emphasized words in the sentences, the named entity's of the words could be used to obtain a better summary.

2.4 Evaluation

Since we do not have human-calculated summaries in hand, we cannot use supervised machine learning models.

As we discussed in the class, ROUGE could be used. Additionally, while I was searching for the 5 informative sentences in those documents, I focused on human, country names, locations, etc. Therefore, it may be possible to enhance the accuracy with NER. Also, since a document may contain different paragraphs that each turns around a different subtopic, it can enhance the similarity if we had first cluster the document and get the most informative 1 or 2 sentence for each, and then concatenate them.