Middle East Technical University | Department of Computer Engineering

CENG 563

Introduction to Natural Language Processing

Fall 2021-2022

Assignment 2

Due: *20.01.2022 Thursday, 23:55*

Instructor: Ayşenur Birtürk
TA: Atakan Garipler

This homework consists of 4 independent tasks. Do not forget that your reports are very important for grading and all questions in the homework text should be properly answered in the report so as to receive full credit. Moreover, there is a bonus task at the end of the document, which can increase your total course grade up to 4 additional points (i.e. it corresponds to an additional 4% to your final grade and it is optional.).

TASK 1: POS Tagging (25 pts)

You are provided formatted data (in conll-u format) and two complete, ready-to-run, Python scripts (one uses Logistic Regression and one uses Conditional Random Fields) for the POS tagging task. Read and try to understand the data and code. Current implementation makes use of a simple feature set, say Set1 (Set1={token, start_with_capital, has_capitals_inside, is_all_capitals, has_numbers}). You will design 2 additional feature sets (Set2 and Set3) and compare performances of 3 feature sets. In order to be able to extract your new features, you will probably have to add a few lines to the already existing code. However, except the *extract_features()* function, its helper, *creatdict()* function and the main, do not make major changes in the given code (some other small changes may be required.). You may add helpers. To be more specific about what to submit:

1. Skimming through the given data, you will see that it consists of three columns. First column is tokens of the sentence. What are the other two columns? Which columns do we use in this task? (Hint: Taking a look to the given code and making a short survey may help.)

2. Run the tagger_lr_pos.py and report the accuracy, precision, recall, F1 values (of Set1). Create 2 new feature sets such that they outperform the given Set1. What are your new feature sets? Report. Also report their accuracies, recalls, precisions and F1 scores. Submit codes using Set2 and Set3 (as two separate Python scripts named tagger_lr_pos_set2.py and tagger_lr_pos_set3.py).

3. Repeat step 2 with tagger_crf_pos.py. (You should also report answer to the same questions, and submit required codes.)

4. Compare two LR and CRF classifiers you have used. Which one is more suitable for this task, why? Report.
   (Hint1: Again, a short survey may help.)
   (Hint2: Note that at the training phase classifiers are fitted sentence by sentence instead of text-by-text etc.)

TASK 2: Named Entity Recognition (25 pts)

Similar to TASK 1, however, the task is NER instead of POS-tagging. Again you will run the existing code which includes feature set 1 and report metrics. Then, again you will create 2 new feature sets specific to this task, report those sets, their metrics and submit your codes. Except the *extract_features()* function, its helper, *creatdict()* function and the main, do not make major changes in the given code (some other small changes may be required.). You may add helpers.

The data you will work on employs IOB encoding. If you do not remember that encoding, starting with remembering it by studying the course material may be helpful.

Training data includes additional tags (i.e. POS-tags) but test data do not. Therefore, at the testing phase, POS-tagger of nltk is used. If you need an additional tagger for this task, use nltk's taggers. If that is the case, you are allowed (and have to) properly add this tagger to the test function. If you do that, indicate in your report.

To be more specific about what to submit:

1. Run the tagger_lr_ner.py and and report the accuracy, precision, recall, F1 values (of Set1). Create 2 new feature sets such that they outperform the given Set1. What are your new feature sets? Report. Also report their accuracies, recalls, precisions and F1 scores. Submit codes using Set2 and Set3 (as two separate Python scripts named tagger_lr_pos_set2.py and tagger_lr_pos_set3.py).

2. Repeat step 2 with tagger_crf_ner.py. (You should also report answer to the same questions, and submit required codes.)

3. What can be done to improve the performance of LR such that it becomes closer to that of CRF? (Hint: Think about the difference between CRF and LR. May adding some type of features reduce the difference?)

4. You will observe that both precision and recall values are remarkably smaller than the corresponding accuracy value. This is due to the testing methodology. What is this testing methodology; why is it needed? (Hint: Investigating the test function and making a short survey will be helpful.)

TASK 3: Dependency Parsing (25 pts)

For this task, you will employ spaCy library of Python. You will work with the data provided by the Universal Dependencies treebanks for Turkish. For more information about the data, you can refer to: http://universaldependencies.org/format.html
https://github.com/UniversalDependencies/UD_Turkish-IMST

Though the task of dependency parsing seems hard to implement at first sight, you will observe that it is not, thanks to spaCy. Refer to the documentation of spaCy in order to find out how to use spaCy for this task: https://spacy.io/api

Submit/report:

1. You may write a Python script or call modules of spaCy one by one from your terminal. Submit your Python script or a txt file that includes calls you have made from the terminal in order.

2. In your reports, explain your implementation **in detail**.

3. Evaluate your final version of the parser on the test set. Report labeled and unlabeled attachment scores (spaCy easily does this.). What are those scores? Is there a remarkable difference between those two scores? Why? Explain in your reports. Do not skip any of the questions.

4. Experiment with your parser: Feed some sentences to your parser, observe correctness of the parsing. Report 1 (almost) totally correctly parsed and 1 poorly parsed sentence. What may cause that difference in the correctness of parsing? Report/discuss.

TASK 4: Extractive Summarization (25 pts)

You are provided with a document collection of Australian legal cases from the Federal Court of Australia (FCA). The document collection includes 4000 legal cases. You may use the whole collection or a restricted number of documents (at least 1000) for this task. If you chose the latter option, please report which documents you have used.

The task of extractive summarization can simply be defined as finding and returning the sentences that are most informative about the document they are contained in. To achieve this, in this specific task, you should calculate ***Tf-Idf weights*** of tokens in the whole document collection. Then for each document, simply calculate ***cosine similarity*** between sentences in document and the remaining of the document (i.e. employ cosine similarity with tf-idf weighting). Then, return most informative 5 sentences as the summary of the document.

Documents are in xml format. Only use sentences in this task (They are tagged with <\sentence> tag.). They are also catchphrases in documents. Do not use them at any stage! Implementations making use of catchphrases will be penalized heavily.

To submit/report:

1. Submit your implementation of the task as a single Python file.

2. Experiment with your summarizer: Read at least 5 documents and summarize them using your summarizer. Are summaries sufficiently informative? How would you summarize those documents? Submit a txt file which includes document names you have used and your personal human-made **extractive** summaries (i.e. select most informative 5 sentences on your own). Is there a big difference between your summary and the automated summary in terms of informativeness? Why? Report.

3. How can this task be enhanced? Discuss. (You may make a short survey about extractive summarization methods.)

4. How can you evaluate such a task? (Again, survey required.)

BONUS TASK:
(Graded seperately, adds up to 4 pts to your total couse grade)

Select ***at most one of the following***. You must inform the TA about your selection until January 6th (Thursday) 23:55 via e-mail (garipler@metu.edu.tr).

Option 1 – Abstractive Summarization:

Design and implement an abstractive summarizer. To achieve that, use vector embeddings form Word2Vec, Glove, BERT etc. together with any algorithm/method (hint: decoder/encoder etc.).

On your reports, you should report your design in detail, justify your design decisions and explain your implementation in detail. You will work on the same dataset as Task4.

Option 2 – Paper Presentation and Summary:

Select a paper published in determined journals/conferences. Write a 1-2 page (abstractive not extractive, generated by yourself not by the computer) summary of that paper. To be more specific, write long abstract (around 2 pages) for that paper in ACL format. Prepare a PowerPoint presentation about the paper and present it in the class, at the last week of lectures. Journals and conferences will be announced later.