INF554 - Machine and Deep Learning
Data Challenge
Extractive Summarization with Discourse Graphs

November 2023

1 Introduction

The goal of this data challenge is to study and apply machine learning techniques to a real-world classification problem. Your mission is to build an *extractive summarization* system that can **binarily predict whether each utterance in a dialogue transcription is important or not** (with label 1 or 0). The set of utterances that are classified as important is called an *extractive summary*.

> **Note**: Summarization approaches fall into two broad categories: *extractive* and *abstractive*. The former creates summaries by directly lifting important sentences from source documents, without any further modifications. The latter involves the generation of more human-like and novel abstractive sentences, via a natural language generation component.

We provide a collection of 137 dialogues originating from role-playing conversations between 4 participants within a fictive electronics company: a project manager (PM), a marketing expert (ME), a user interface designer (UI), and an industrial designer(ID).

For each of the dialogues, you have access to two types of data:

1. its transcription: a sequence of utterances (spoken sentences in text and the speakers).
2. its discourse graph: a directed weakly connected graph reflecting the discourse structure, in which the nodes represent utterances and the edges represent discourse relations (e.g., elaboration, clarification, completion).

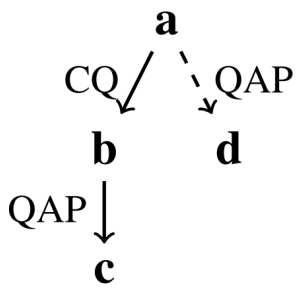

Figure: Discourse graph for the example:
a. Alex: What is Mary's last name?
b. Bill: The Mary from our reading group?
c. Alex: Yes.
d. Bill: It's Johnson.
The dashed edge represents a long distance link. CQ and QAP stand for 'Clarification Question' and 'Question/Answer Pair'.

This setting poses several challenges ranging from data preprocessing to model design and hyperparameter tuning. Your solution can be based on supervised and/or unsupervised learning techniques for the text and the graph, i.e., natural language processing (word embeddings,

language models, etc.) and graph machine learning (graph features, graph neural networks, etc.). You should aim for a **maximum F1-Score**.

This data challenge is hosted on Kaggle as a competition. In order to access the competition, you must have a Kaggle account. If you do not have an account you can create one for free. The URL to register for the competition and have access to all necessary material is the following:

https://www.kaggle.com/t/d222accd5e424ba6826852c0cd2f3893

2 Dataset and Baseline Description

We have in total 97 and 40 dialogues for training and test respectively, corresponding files are placed under the `training` and `test` folder.

For each dialogue, with ID such as `ES2002a`, there are two files:

ES2002a.json - the dialogue transcription: a list of utterances, each of which consists of speaker, text, and index.

```
[
    ...,
    {
        "speaker": "PM",
         "text": "Um so first of all , just to kind of make sure that we all know each other ,",
        "index": 5
    },
    {
        "speaker": "PM",
        "text": "I'm Laura and I'm the project manager .",
        "index": 6
    },...
]
```

ES2002a.txt - the discourse graph: an edge list with edge attribute, indicating the discourse relation. (e.g., the utterance of index 5 and the utterance of index 6, are connected with one edge, of type `Elaboration`)

```
...
4 Continuation 5
5 Elaboration 6
6 Continuation 7
...
```

For the training set, the binary target labels of all dialogues can be found in **training_labels.json**. (e.g., for the utterances of index 0, 1, 2, … of the dialogue ES2002a, their labels are [0, 0, 1, …])

```
{
...
    "ES2002a": [
        0,
        0,
        1,
        ...
    ],
...
}
```

The objective is then to firstly train a system / model with the above training data, secondly predict the labels for all the dialogues in the test set, and create a test_labels.json file gathering the predictions.

Running the **baseline.py** will generate the results of two baseline systems: `test_labels_naive_baseline.json` and `test_labels_text_baseline.json`.

You will need to install the following dependencies (potentially on the basis of the anaconda python distribution):

```
pip install jsonargparse sentence-transformers
```

In the end, convert the results in json format to kaggle-compatible `submission.csv` with:

```
python make_submission.py --json_path test_labels_naive_baseline.json
```

Now, it's ready to upload the `submission.csv` to Kaggle.
We provide a file **submission_naive_baseline.csv** for the reference.

3 Task and Evaluation

Your model should predict whether an utterance is a part of the extractive summary (1) or not (0). As the classes are imbalanced, the evaluation metric for this competition is the F1-Score (as implemented here, with default parameters), over all the utterances of the entire test set.
The F1 score is a measure of accuracy that averages precision and recall. Precision is the ratio of true positives (tp) to all predicted positives (tp+fp), while Recall is the ratio of true positives to all actual positives (tp + fn). The F1 score is given by :

$$F1 = 2\,\frac{pr}{(p+r)}\ where\ p = \frac{tp}{(tp+fp)},\ r = \frac{tp}{(tp+fn)}$$

4 Useful Python Libraries

In this section, we briefly discuss some packages that can be useful in the challenge:

- A very powerful machine learning library in Python is scikit-learn[1]. It can be used in the preprocessing step (e.g., for feature selection) and in the classification task (several algorithms have been implemented in scikit-learn).
- A very popular deep learning library in Python is PyTorch[2]. The library provides a simple and user-friendly interface to build and train deep learning models.
- Since you will deal with textual data, the Natural Language Toolkit (NLTK)[3] of Python can also be found useful.
- Gensim[4] is a Python library for unsupervised topic modeling and natural language processing, using modern statistical machine learning. The library provides all the necessary tools for learning word and document embeddings. An alternative to it is FastText[5].
- If you do not want to spend a lot of time producing the word embeddings, you can use transfer learning. You can download a pretrained set of word embeddings on GoogleNews[6], Glove[7] or Numberbatch[8] - additionally, while averaging the embeddings of each word in an utterance is a valid representation of a sentence, the *Sentence Transformer*[9] library allows you to use transformer based representation of utterances.
- In case you want to work with the data represented as a graph, the use of a library for managing and analyzing graphs may prove to be important. An example of such a library is the NetworkX[10] library of Python that will allow you to create, manipulate and study the structure and several other features of a graph.
- Additionally, using Graph Neural Networks can help for node classification. In case you want to use GNNs, the Pytorch Geometric[11] library provides a way to use most common GNNs architectures in a straightforward fashion.

5 Grading Scheme

Each team has to be composed of 2 or 3 students. No larger or smaller teams will be accepted. Please join and use the slack channel (”team-finding”) to organize yourselves into groups or coordinate offline. Group choices need to be submitted to us by **Friday 24th November at 17:00** at the below link:

---

[1] http://scikit- learn.org/
[2] https://pytorch.org/
[3] http://www.nltk.org/
[4] https://radimrehurek.com/gensim/
[5] https://fasttext.cc/
[6] https://code.google.com/archive/p/word2vec/
[7] https://nlp.stanford.edu/projects/glove/
[8] https://github.com/commonsense/conceptnet-numberbatch
[9] https://www.sbert.net/
[10] http://networkx.github.io/
[11] https://pytorch-geometric.readthedocs.io/en/latest/

https://forms.gle/Nx4AJMzo7n65xJP1A

Grading will be out of 100 points in total. Each team should deliver:

A submission on the Kaggle competition webpage. (20 points) will be allocated based on raw performance only, provided that the results are reproducible. That is, using only your code, the data provided on the competition page and *any additional resources you are able to reference and demonstrate understanding of*, the jury should be able to train your final model and use it to generate the predictions you submitted for scoring.

A zipped folder in moodle (30 points) including:

1. A folder named ”code” containing all the scripts needed to reproduce your submission.
2. A README file with brief instructions on how to run your code and where it expects the original data files.
3. A report (.pdf file), of max 3 pages, excluding the cover page and references. In addition to your self-contained 3-page report, you can use up to 3 extra pages of the appendix (for extra explanations, algorithms, figures, tables, etc.). Please ensure that both your real name(s) and the name of your Kaggle team appear on the cover page.

The 3-page report should include the following sections (in that order):

- **Section 1: Feature Selection/Extraction (10 points).** Independent of the prediction performance achieved, the jury will reward the research effort done here. Best submissions will capture both graph and text information. You are expected to:
    1. Explain the motivation and intuition behind each feature. How did you come up with the feature (e.g., are you following the recommendation of a research paper)? What is it intended to capture?
    2. Rigorously report your experiments about the impact of various combinations of features on predictive performance, and, depending on the algorithm, how you tackled the task of feature selection.
- **Section 2: Model Choice, Tuning and Comparison (15 points).** Best submissions will:
    1. Compare your model against different algorithms (e.g., Neural Network, Support Vector Regression, Random Forest...).
    2. For each algorithm, explain the procedure that was followed to tackle parameter tuning and prevent overfitting.
- Report and code completeness, organization and readability will be worth **5 points**. Best submissions will (1) clearly deliver the solution, providing detailed explanations of each step, (2) provide clear, well-organized and commented code, and (3) refer to research papers. You are free to search for relevant papers and articles and try to incorporate their ideas and approaches into your code and report as long as (a) it is clearly cited within the report, (b) it is not a direct copy of code and (c) you are able to demonstrate understanding of the content you incorporated.

An oral presentation of your project and the achieved results (50 points) Oral presentations will be scheduled in the week of the 11th or 18th of December. A schedule on which you can choose presentation slots will be released after the team submissions have been made. We ask you to prepare 15-minute presentations, which will then be followed by questions from the examiners. It is required that all group members are present and take a speaking role during the presentation.

7 Submission Process

The competition ends on **Friday 8th December at 17:00**. This is the deadline for you to submit a compressed file containing your source code and final report, explaining your solutions and discussing the scores you have achieved. Until then, you can submit your solution to Kaggle and get a score at most 5 times per day. There must be one final submission per team.