INF554 - FINAL REPORT

Extractive Summarization with Discourse Graph

December 8th, 2023

Gaagle AI - Raphael Manus, Darius Dabert, Jules Peyrat

0.1 Model Overview

The table below shows all of the different extractive summarization configurations that we have tried. They are all based on an encoding technique (whether we encode the whole graph or individual sentences), an encoding model used to generate embeddings of the encoded object and a classification model which takes as input the previous encoding. We indicate a rough estimation of the performance of this model on a validation dataset taken as 20% of the provided training set.

Encoding Model	Encoding Technique	Classification Model	Validation Performance
all-MiniLM-L6-v2	Sentence-Based	Decision Tree	38%
all-MiniLM-L6-v2	Sentence-Based	MLP	57%
all-MiniLM-L6-v2	Sentence-Based	Random Forest	25%
all-mpnet-base-v2	Sentence-Based	MLP	57%
all-MiniLM-L6-v2	Sentence + Annotations	MLP	58%
all-MiniLM-L6-v2	Full Graph	GNN	60%
all-MiniLM-L6-v2	Full Graph	GNN + MLP + Vote	63%

0.2 Simple Improvements to the baseline

We started off with the provided baseline model which uses a sentence transformer to generate encodings, and a decision tree to classify individual sentences.

We tried **switching to a stronger encoder** hoping that better sentence embeddings would make the classification model perform better overall. We switched to *all-mpnet-base-v2* which is supposed to have slightly better encoding performance according to the official website, however we didn't notice a difference with MLP and Decision Trees as classification models. As the model was 4 times heavier and significantly slower, we used the provided MiniLM for all future experiments.

We also tried **including annotation data from the discourse graph to the individual sentences**. For instance, if a given utterance is an answer to a previous utterance, then the model would encode *[speaker] answers on [previous utterance] : [actual utterance]* instead of the default *[speaker] : [utterance]*. This yielded slightly better results specifically on MLPs. Although there are definitely other ways of including local discourse graph data through plain text, like passing the n last utterances to the encoding model instead of only one utterance, we have not experimented with these techniques.

We also tried **switching to other classification models**. We first tried using a Random Tree classifier instead of the baseline Decision Tree, which significantly yielded worse results. Moreover, as BERT-like encoders are usually used along with simple feed-forward neural networks, we tried using a simple MLP with two layers. We could rapidly reach a validation performance of 57% outperforming baseline by about 20%.

0.3 Using a sentence-based MLP classifier

Simple MLPs yield very decent performance at very low cost (our models weighing less than 1MB). We settled with 3 hidden layers as the model was unable to efficiently train more, and less would yield sub-optimal results. MiniLM provides 386-dimensional vectors, so we chose 256, 32 and 1 for layer dimensions as it seemed to yield optimal performance. We alternated between feedforward and ReLU layers as suggested by recent work. We added a dropout layer with probability 0,3 which seemed to be an optimum, and a sigmoid layer as last layer for classification.

The features we gave to MLPs are encoded individual sentences, including the speaker and the utterance type provided by the discourse graph (is it an answer, continuation of a previous utterance ?), for example *PM answers : I don't think so uhhh.*

We trained these models for 40 epochs using cross entropy as loss function and Adam optimizer with a learning rate of 0,003. This value for learning rate reduced oscillations in the beginning of the training phrase. We also used a plateau scheduler for learning rate which lowers the learning down to 10% of its original value when reaching 20 epochs, which helped the models converge.

We trained multiple models and saved checkpoints at every epoch. We plotted graphs (see section below) to monitor validation loss and performed early stopping in order to avoid over-fitting the training data. We used a batch size of 8 for training and validation as it seemed to be an optimum.

0.4 Using Graph Neural Networks

The provided discourse graphs encouraged us to use graph/dialogue structure in order to provide more context to the classification model, hopefully increasing overall performance. To do so, we used the PyTorch Geometric library to both build graph data based on provided dialogue information, and train graph neural networks using predefined layer architectures.

We also provided One-Hot Encodings of the discourse graph edge types (answers, continuations, clarifications...) using the "edge attributes" feature. This can help the model picking up on important utterances.

We used the same training techniques (scheduler, loss function, optimizer, epoch number, early stopping, batch size) as with MLPs. We tried different architectures for our graph neural networks, and found out that the best performing models are those using predefined graph convolutional layers, followed by multiple classical MLP layers. For instance, we obtained out best performing model with 2 4096 and 2048-dimension ResNet layers, followed by two regular MLP layers of dimensions 2048 and 256 (with ReLUs + dropout). We had to use a much higher dimension number than with MLPs to obtain similar/better performance as full graphs contain much more information than individual utterances.

Optimizing hyperparameters on GNNs was much more difficult than optimizing MLPs, and GNNs yielded much higher variance in terms of model performance. Figure 1 and 2 presents training curves of two models that differ with the use of either TransformerConv or GCN prefined GNN layers.

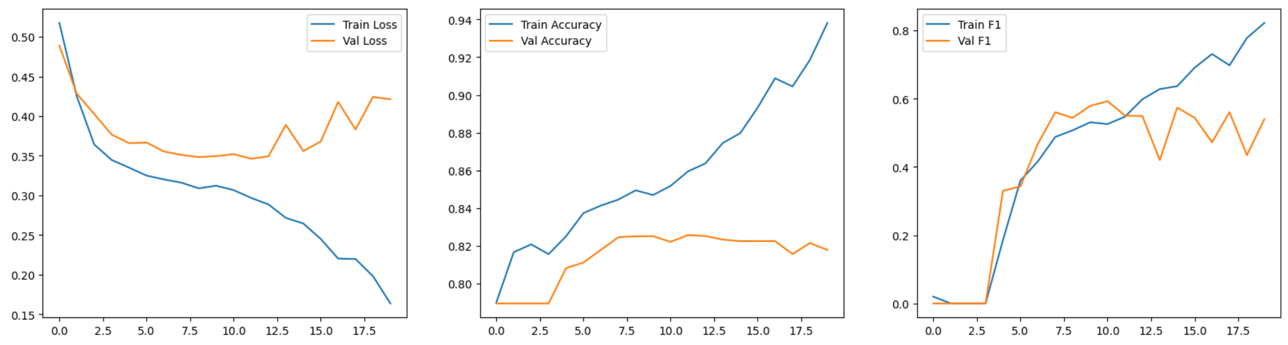

Figure 1 - Well performing GNN (TransformerConv-Based), stopped early at epoch 10

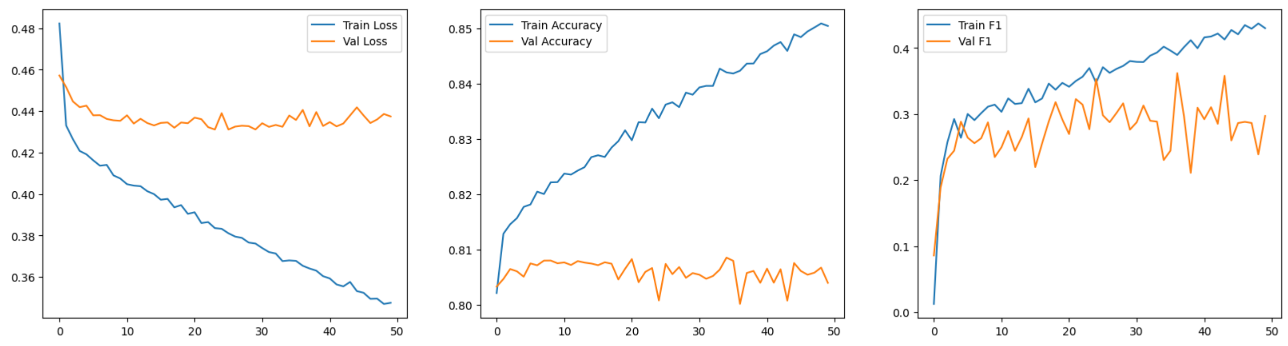

Figure 2 - Overfitting GNN (GCN-Based) with poor performance

0.5 Combining models with "Majority" Vote

To boost our model's accuracy and make predictions more consistent, we combined results from various models, each trained on different splits. During cross-validation, these models encountered different data and used diverse settings and architecture, gaining unique insights into various aspects of the information. We kept our 3 best MLP-based models and our 3 best GNN-based models for a total of 6 models.

By aggregating their predictions, we aimed to get a more reliable assessment of whether a given statement was crucial. We established a threshold, a choice made using a part of the training set not seen during cross-validation, to decide when to classify a sentence as '1'. If a sufficient number of models predicted '1' for a statement, indicating a consensus, we classified it as such. This approach resulted in a notable three-point improvement in the F1 score of the combined models. This not only boosted the F1 score but also ensured a stronger and more dependable model by incorporating diverse information learned by each model during training.