INF554

Machine Learning and Deep Learning
Fall 2023

Data Challenge Report

Prepared by:

Student Name	Kaggle Username	Email Address
Antonio Nisi	antonionisi	antonio.nisi@polytechnique.edu
Mirette Moawad	miretteamin	mirette.moawad@polytechnique.edu
Nader Khalil	naderyouhanna	nader.khalil@polytechnique.edu

I. Feature Selection & Extraction:

1. **Data Cleaning:**

Before converting the sentences to numerical features, we clean the data by removing stopwords. For this we have used the NLTK stopwords.
We also manually remove every instance of <*vocalsound*> in the dataset.
We noticed that including the speaker with the sentence does not significantly change overall performance.

2. **Sentence Embedding:**

In order to generate the features that we feed into the neural network, we embed each sentence in a high dimensional space. Since the output is classification of sentences, it is more logical to use sentence embeddings rather than word embeddings.
We have tried the following sentence embeddings:

- Bert
- USE
- TF-IDF

And have found the bert embedding to give the best results.
Specifically, the final model uses the all-MiniLM-L6-v2 sentence transformer.
Sentence models (like sentence bert) do some sort of pooling over the token representations to output a 1-d vector for each sentence.
This pooling might use the CLS token, mean or max operations.
The premise is: this general purpose pooling might not be so effective when doing extractive summarization.
We test that by learning the pooling over tokens with attention pooling hopefully that will customize the sentence representation for the task.
After encoding the data, we normalize it using StandardScaler from sklearn.preprocessing.
We also tried adding a normalization layer in the GNNs themselves, but this did not change overall performance.

3. **Feature selection:**

We have tried applying PCA to the features we get out of the embedder. However, since the dimensionality of the feature space is already small (384 features), we have found that PCA was not beneficial to the final result, so we did not keep it in the final model.

4. **Knowledge Graph:**

The sentence embeddings capture the text information, and the relations graph is constructed in order to capture the discourse graph information. In the relations graph, nodes represent sentences while edges represent the sentence relationships to each other (Continuation, Explanation, etc…)

We have only 1 type of nodes which is the sentence node and 16 types of edges that are embedded by PyTorch geometric by assigning their edge indices *(Figure 1)*. This graph is a heterogeneous graph as we have multiple types of edges.

5. **Train /Validation split:**

We split the data into a validation set *(20%)* and a train set *(80%)*

II. Model Choice, Tuning and Comparison

1. **Model:**

We have opted for GNN's using Pytorch Geometric.
We tried using multiple neural networks:

- GraphSAGE
- GraphGAT
- GraphGATv2
- GraphGIN
- GraphGCN

And we settled for a model that votes between the predictions generated by:

1. GraphSAGE
2. GraphGATv2
3. GraphGIN
4. GraphGCN

2. **Model Selection:**

Other models we have tried include:

- Random Forests using RandomForestClassifier module from sklearn.ensemble
  - F1 score: 0.316
- Catboost using CatBoostClassifier module from catboost (provided text baseline)
  - F1 score: 0.395
- SVM using svm module from sklearn
  - F1 score: 0.418
- One layer GNN:
  - F1 score: 0.53

3. **Hyperparameters:**

The hyperparameters included:

- Learning rate
- Number of epochs

- Dropout
- Batch Size
- Weights of classes

We tuned the hyperparameters manually and have settled for:

- Learning rate = 10^{-4}
- Epochs = 3000 epochs
- Batch size = 8
- Dropout values = 0.3
- Weights of classes = ratio of 1 to 5, with the higher weight given to the ‘1’ class. This distribution corresponds to the class distribution in train data *(Figure 2)*

III. Appendix

Explanation of some of the concepts and algorithms used:

1. **GNN:**

Graph Neural Network (GNN) *(Hong, 2020)* is categorized as a neural network that is specifically designed to work with graph-structured data. They can do what Convolutional Neural Networks (CNNs) failed to do as they are designed to work with non-grid-like data which is powerful for dynamic graphs with non-fixed size unlike CNNs, and are particularly useful for tasks that involve analyzing and making predictions about complex, interconnected systems. One of the key strengths of GNNs is their ability to handle relational information among the nodes in the graph; they can incorporate the relationships between different entities into their predictions. In the knowledge graph, nodes represent the entities in the system and edges represent the relationships between those entities.

- There are various approaches to perform neighbor's aggregation (also referred to as message passing), the ones we have tested:

  a. **GraphSAGE:**
  This technique as explained in paper *(William L. Hamilton, 2018)* is an inductive representation learning approach, but its message passing method is applicable to all GNN tasks. *(Figure 4)* It samples nodes from the local neighborhood of the target node, and it performs multiple aggregation functions on those sampled node embeddings, those aggregators include:

  - Mean Aggregator:

$$h_v^k \leftarrow \sigma\left(W \cdot MEAN\left(\left\{h_v^{(k-1)}\right\} \cup \left\{h_u^{(k-1)}, \forall u \in N(v)\right\}\right)\right)$$

    Where h_v^k is the target node, $N(v)$ is the neighborhood of h_v^k and W is a learnable matrix.

  - Max Pooling Aggregator:

$$AGGREGATE_{pool}^k = \max\left(\left\{\sigma\left(W_{pool} h_{u_i}^k + b\right), \forall u_i \in N(v)\right\}\right)$$

  b. **Graph Attention Networks (GAT):**
  This technique as explained in paper *(Bengio, Liò, Romero, Casanova, & Cucurull, 2018)* basically performs a weighted average of neighboring node embeddings, and each node weight is calculated using a dot product between the project representation of the target node and the neighboring node. Then, the attention weights are passed through a SoftMax function, which ensures that the weights are normalized.

$$e_{ij} = a\left(Wh_i, Wh_j\right) \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})}$$

$$h_i' = \sigma(\sum \alpha_{ij} * Wh_j)$$

IV. Figures:

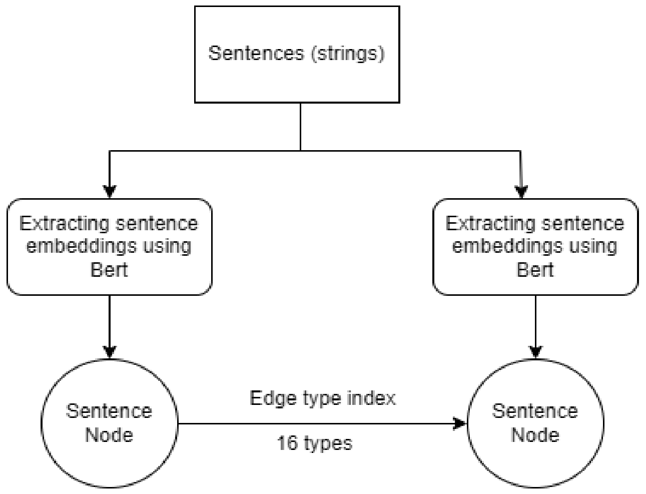
Sentences (strings)
Extracting sentence embeddings using Bert
Extracting sentence embeddings using Bert
Sentence Node
Edge type index
16 types
Sentence Node

Figure 1 Knowledge Graph

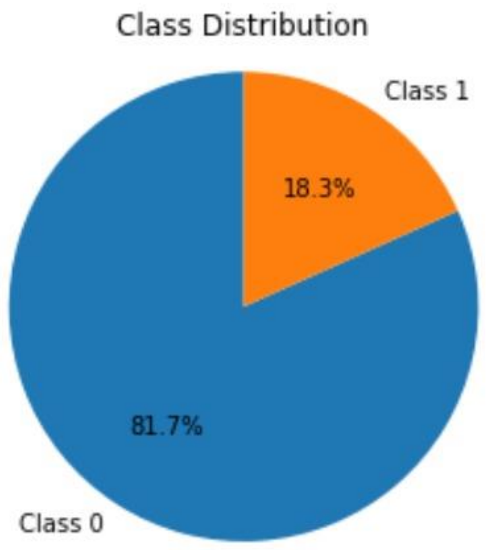

Figure 2 Class distribution

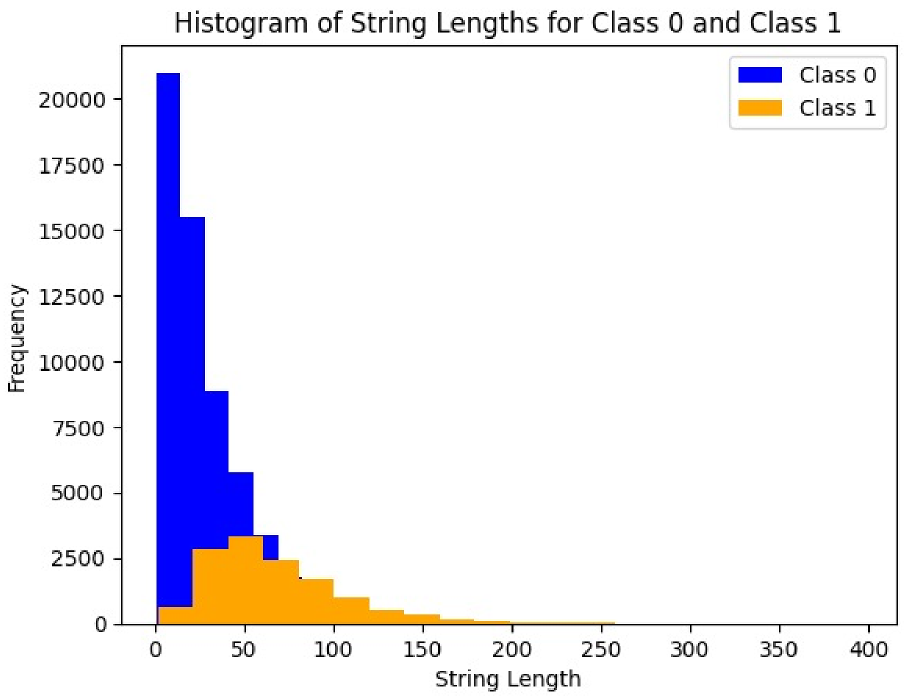

Figure 3 Class distribution with respect to sentence length

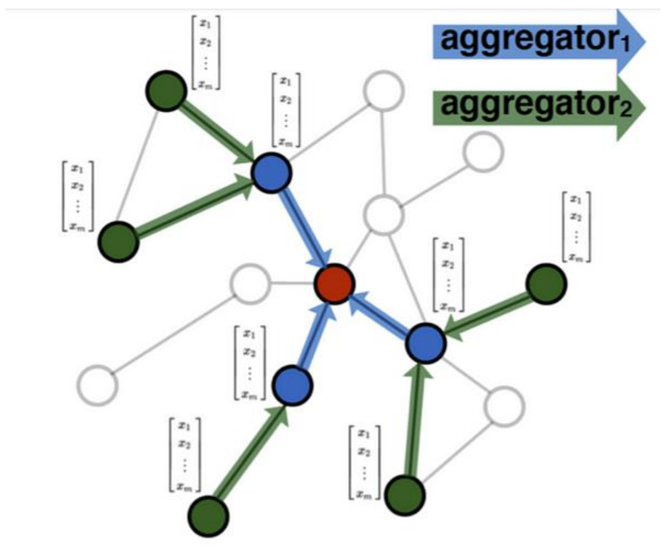

Figure 4 GraphSAGE message passing

V. References:

- William L. Hamilton, Rex Ying, Jure Leskovec, (2018) “Inductive Representation Learning on Large Graphs”
- Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, Yoshua Bengio, (2018) “Graph Attention Networks”
- Shaked Brody, Uri Alon, Eran Yahav, (2021) “How Attentive are Graph Attention Networks ?”
- Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka,(2018) “How Powerful are Graph Neural Networks ?”
- Hong, S., (2020) “An Introduction to Graph Neural Network(GNN) For Analysing Structured Data”