Data Challenge

INF554

Axel Benyamine
Louis Collomb
Othman Hicheur

Introduction

Potential uses

- Crime prevention spying software
- Better share business information within companies
- Create summaries after meetings

Kaggle team : **Garbage Collectors**

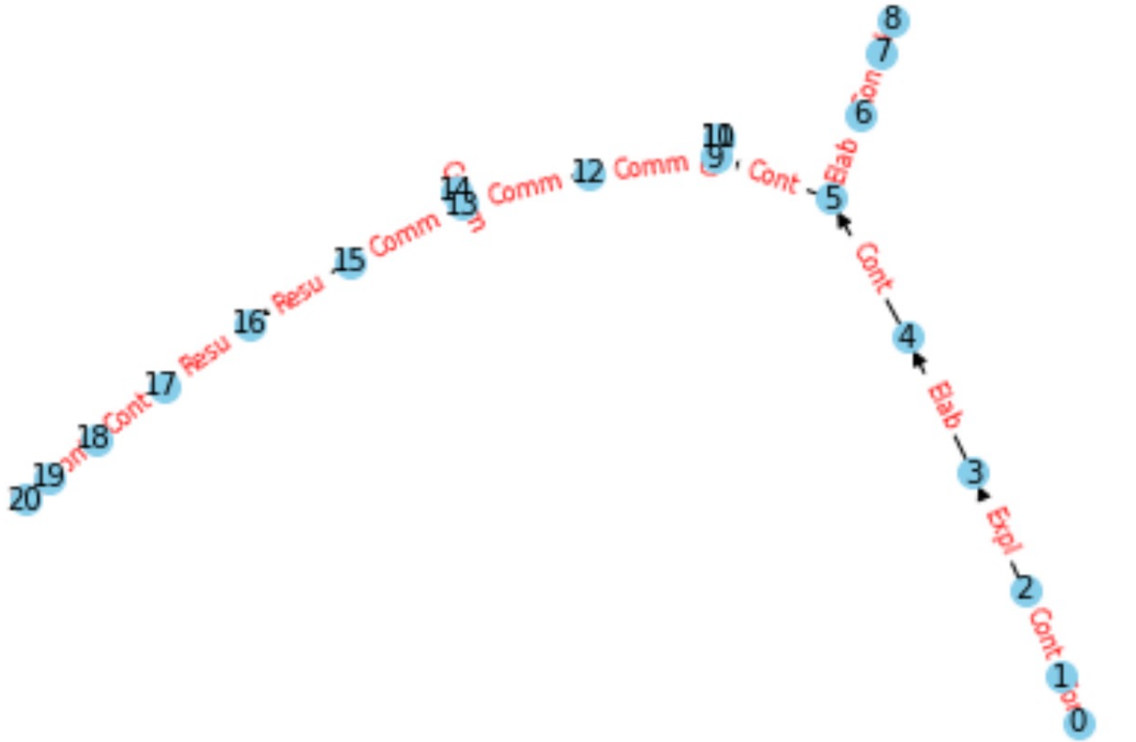
Table of contents

I - Preprocessing

Information graph

Informations

- Graphs representing the conversations
- No, one or multiple exit links
- A nametag for each link



Our encoding

TF tokenizer + embedding

- Use of padding after tokenizing
- Using word tokenizer and word embeddings

Keeping the index

- The index of the speaker is encoded with the sentence

Bert encoding

- Sentence Transformer all-MiniLM-L6-v2

Links between sentences

- Stats between the incoming and exit links to other sentences

Incoming links

- We look for :

$$\mathbb{P}\,(\,\mathrm{B} = 1 \mid \mathrm{A} \rightarrow \mathrm{B} = \mathrm{type}) = \frac{\sum_{(A,B)\,\mid\, A \rightarrow B = type} f(B)}{\sum_{(A,B)\,\mid\, A \rightarrow B = type} 1}$$

- Average probability : 0.18

- Some links very rarely lead to information and others very often lead to information

$\mathbb{P}$	**Best probabilities**	$\mathbb{P}$	**Worst probabilities**
0.32	‘Elaboration’	**0.02**	'Acknowledgement'
0.29	'Continuation'	**0.04**	No incoming link
0.28	Contrast'	**0.05**	'Correction'
0.26	'Explanation'	**0.08**	'Comment'

Exit links

- We look for :
- $$\mathbb{P}\,(\,\mathrm{A}=1\mid \mathrm{A}\to \mathrm{B}=\mathrm{type})=\frac{\sum_{(A,B)\mid A\to B=type} f(A)}{\sum_{(A,B)\mid A\to B=type} 1}$$

- Average probability : 0.18

- Less of a clear-cut on the links

$\mathbb{P}$	**Best probabilities**	$\mathbb{P}$	**Worst probabilities**
0.34	‘Alternation’	**0.07**	No exit link
0.34	’Parallel'	**0.14**	'Narration'
0.30	‘Continuation'	**0.20**	'Correction'

II – Initial models experimented

- Initial idea : use Bert encoder with speaker and the associated sentence as initial :

```
X_training.append(utterance["speaker"] + ": " + utterance["text"])
```

```
X_training = bert.encode(X_training, show_progress_bar=True)
```

- After, we tried to train some machine learning models from scikit-learn like MLPClassifier :

```
clf = MLPClassifier(random_state=1, max_iter=1000, hidden_layer_sizes=(1000), activation='relu', solver='adam')
clf.fit(X_new_training_preprocessed, y_new_training)
```

We decided to take the parameters hidden_layer_sizes, activation, solver and max_iter for instance because it seems to be the best for a grid-search

- We also tried the SVM classifier from scikit-learn :

```
clf = SVC(kernel='rbf', C=1, gamma='auto', random_state=42, max_iter=1000)
clf.fit(X_new_training_preprocessed, y_new_training)
```

We chose this model because we thought that SVM was a good classifier for this type of problem. Nevertheless, with the high complexity of the datas, SVM did not manage to separate datas.

- XGBoost was an other classifier from scikit-learn :

```
clf = XGBClassifier(n_estimators=1000, learning_rate=0.01, max_depth=10, random_state=42)
clf.fit(X_new_training_preprocessed, y_new_training)
```

We tried this model as we seen in the course. The results was better than SVM.

There are the parameters for the grid-search :

However, the results for the F1-score were not as good as we expected

```python
param_grid_mlpc = {
    'hidden_layer_sizes': [(100,), (1000,), (100, 100), (1000, 1000)],
    'activation': ['relu'],
    'solver': ['adam', 'sgd'],
    'max_iter': [200, 500, 1000],
    'random_state': [42]
}

param_grid_svm = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto'],
    'C': [0.1, 1, 10, 100, 1000],
    'max_iter': [1000],
    'random_state': [42]
}

param_xgboost = {
    'n_estimators': [100, 500, 1000],
    'max_depth': [3, 5, 10],
    'learning_rate': [0.01, 0.1, 0.5],
    'random_state': [42]
}
```

```
for utterance in transcription:
    X_training.append(utterance["text"])
    speaker_ids = np.append(speaker_ids, utterance["speaker"])

for i in range(len(speaker_ids)):
    if speaker_ids[i] == 'UI':
        speaker_ids[i] = 0.0/6
    elif speaker_ids[i] == 'PM':
        speaker_ids[i] = 1.0/6
    elif speaker_ids[i] == 'ID':
        speaker_ids[i] = 2.0/6
    elif speaker_ids[i] == 'ME':
        speaker_ids[i] = 3.0/6

speaker_ids_list.append(speaker_ids)
X_training = bert.encode(X_training, show_progress_bar=True)
y_new_training += training_labels[transcription_id]
X_new_training.append(X_training)
```

After these tries, another idea was to change the preprocessing of the datas. We only put the sentences in the training datas and we encoded them, after we add the speaker in a float shape at the end.

The idea to separate the speaker from the sentence was to not lose the information from the person who is speaking in the conversation, which seemed to be important

From all these experiments, the best results went from the MLPClassifier from scikit-learn all the time. That’s why we decided to create our proper **classical neural network**

III – Other models using Neural Networks

III – Models using **Neural Networks**

- Classical **NN (Feedforward)**
- **RNN** using an **LSTM Layer**
- **Graph Convolutional Networks**

Other models using **Neural Networks**

Classical **NN (Feedforward)**

Structure

- An input layer, one or more hidden layers and an output layer
- Information flows in one direction, without cycles or loops

Characteristics

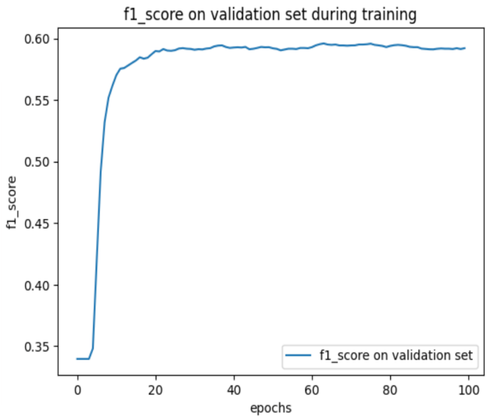
- Linear layers (and dropout)
- Activation function : ReLu
- Output activation function : Sigmoid (because of binary classification)
- BCE Loss with Adam optimizer

Parameters

	After grid search
Number of layers and hidden size(s)	2 layers (50)
Learning rate	10^{-4}
Dropout threshold	No dropout
Threshold for binary classification	$p_{mean} = 0.18$

Advantages of the model

- Very good understanding of all the hidden processes
- **Best f1 score (val/test): 0.58-0.59**




```
final f1_score of validation 0.5921
```

Other models using **Neural Networks**

RNN using an **LSTM Layer**

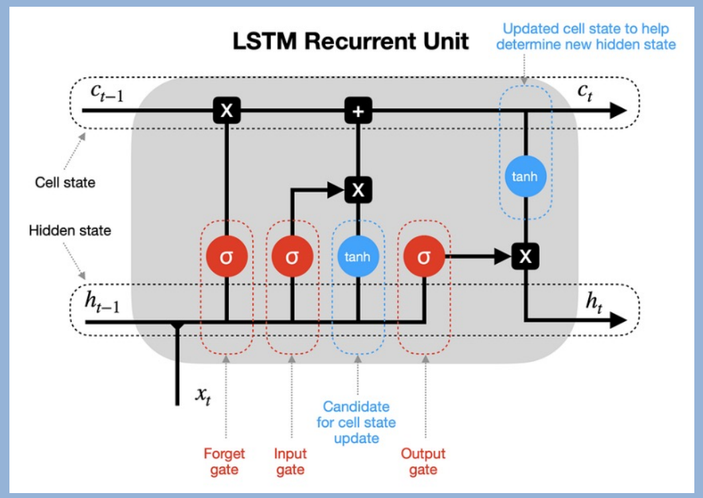
Sentence Transformation :

- Bert

or

- Tokenizing, Padding, Embeding

Recurrent layer



Output activation function :

- Sigmoid

Structure

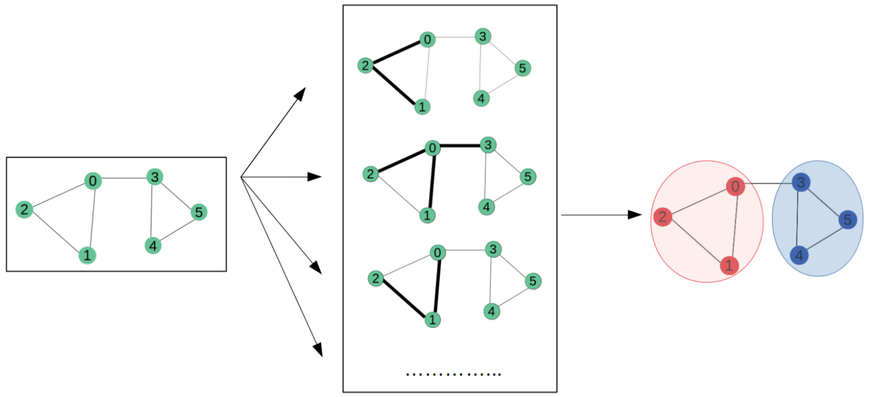
- Incorporates recurrent connections, allowing the network to maintain a hidden state that captures information from previous time steps
- Suitable for processing sequential data with varying lengths

Best results obtained :

- Bert transformation, single LSTM (128), Sigmoid
- Normal preprocessing : f1 score = 0.49
- **Link preprocessing : f1 score = 0.53**

Other models using **Neural Networks**

Graph Convolutional Network



Structure	Preprocessing adaptation	Difficulties faced	Results
• Captures and leverages the inherent graph structure, allowing for seamless processing of interconnected data • Enabling superior performance in tasks involving relational dependencies among entities	• Creation of a NetworkX graph • Extraction of the adjacency matrix • Normalization of adjacency matrix	• Necessity to batch by conversation (RAM limits) • Keeping the link types in the adjacency matrix	• Relatively poor results : f1 score = 0.49 • Our understanding : misinterpretation of certain links by the model (which wasn't aware of the different link types)

Conclusion

Conclusion

Importance of preprocessing • The importance of solid foundations to prevent time losses • Analyzing graph structure and link types to add more information to the input • Possibilities for sentence transformation	**Many degrees of freedom in the design of a model** • Variety of models that could be adapted to this problem • Fixing all the parameters
Failure teachings • Re-thinking the preprocessing • Trying other implementations of the model • Trying other ways of approach	**If we had to continue the project ...** • Implementing link preprocessing on Feedforward NN • Focus on GCN • Extending the GCN to adjacency matrix with categorical values

Thank you for listening !

Bibliography

XGBoost:

Tianqi Chen, Carlos Guestrin, "XGBoost: A Scalable Tree Boosting System," 2016, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

---

Recurrent Neural Network (RNN):

Sepp Hochreiter, Jürgen Schmidhuber, "Long Short-Term Memory," 1997, Neural Computation

---

Graph Convolutional Network (GCN):

Thomas N. Kipf, Max Welling, "Semi-Supervised Classification with Graph Convolutional Networks," 2017, International Conference on Learning Representations (ICLR)