Extractive Summarization with Discourse Graphs: A Kaggle Project on Identifying Key Messages in Business Dialogues

Samuel Gaudin, Mathias Grau, and Alexandre ver Hulst
École Polytechnique

In the Kaggle project we aimed at identifying key messages in professional dialogues, we explored diverse approaches, focusing on specific vocabulary and advanced machine learning techniques. Our dataset included varied business conversations, analyzed for crucial keywords and links using Natural Language Processing techniques. We experimented with traditional algorithms like logistic regression and XGBoost, alongside deep learning, employing multiple neural networks. These models were trained to discern not just the text but also the context and significance of dialogues. Despite challenges like contextual ambiguity and complex business jargon, our approach showed promising results in streamlining professional communication and enhancing decision-making processes.

Feature Selection/Extraction

Introduction to project

The data provided to us is a collection of 137 (97 for training) dialogues between 4 protagonists. We also know the causal links between messages (continuation, elaboration, explanation, etc.). The aim is to train a model capable of accurately predicting the importance of a message.

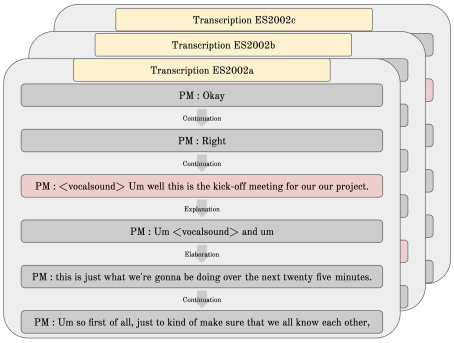


FIG. 1. Succession of transcriptions with labeled messages (grey refers to non important message and red refers to important ones)

Feature set expansion

In this section, we will present all the syntactic, lexical, technical and contextual features we have considered in order to predict the importance of a message (see appendix for more detail).

a. Messages size: In terms of macroscopic consideration, we have considered message size as a factor of importance. In fact, it seemed quite likely to us that a long sentence was more likely to be labelled important than a sentence of average length. The length histograms (appendix) confirmed this tendency, although this criterion is not totally decisive.

b. Type of speaker: We then considered taking into account the type of speaker (PM, UI, etc.). Indeed, it was coherent to assume that the project manager was more likely to deliver important messages than his colleagues, or vice versa. However, as confirmed by the breakdown of important messages by speaker (18% for each), this criterion should unfortunately not be considered as absolute. So we included a feature that identifies the individual since certain speakers might consistently convey more critical content.

c. Sentence embedding: So as to capture the meaning of messages as accurately as possible, we started with a study of different sentence embeddings using BERT, Word2Vec and PyTorch Embedding. We have tried out these embeddings with considerations for sentence cleaning (lower, stop words, non-alphabetics, etc.). The specificity of Word2Vec was that the embedding was an embedding by word, which meant that we had to transform it into a vector for each text of a dialogue by ourselves. We have tried several methods, but the best in our eyes remained the maximum, in norm, of the vector of all words in a text. The two other methods were really efficient, and the results were similar when using either of those 3 methods. Practically, we mainly used BERT and Word2Vec when making our tests, and the best embedding to use depended on the machine learning method used. To see more specifically the benefits of each method, we have detailed our researches in the appendix section.

d. TF-IDF score (importance of words): Then, we analysed the TF-IDF feature (which vectorizes each word), under the assumption that high TF-IDF words score carry more significance and could be indicative of important messages within the global context of our dataset. In our case, we took the maximum vector of each text to add it to a new column of our dataframe. The rarity of terms was expected to correlate with specificity and importance in professional dialogues.

e. TF-IDF score per speaker: Additionally, we explored a TF-IDF variation by speaker, which aimed to capture individual language patterns that might signal importance. In this case, the goal was to underline a significant change of vocabulary when saying important phrases, which can be specific to each individual. Furthermore, we used the "previous" and "next" message links provided in the dataset as features to understand the contextual flow of conversation.

f. Sentiment analysis: We implemented a sentiment analysis via NLTK's sentiment analyzer added a nu-

anced layer, distinguishing messages by their emotional tone. We posited that messages with extreme sentiments (highly positive or negative) are more likely to be significant than neutral ones, which could represent routine or less informative exchanges.

g. New specific word: A novel technical aspect was the introduction of a "new word" feature—assigning a value if a message contained a new word not ranked among the 300 most common words, providing insight into the appearance of innovative or less frequent ideas within dialogues. In this case, we supposed that new words were used more frequently in dialogues that had more importance.

h. Sentence importance score: Lastly, we computed a "phrase importance" score, contrasting the frequency of words in significant versus non-significant messages. This metric assigned values based on whether a word was prevalent in important messages, unimportant ones, or neither.

Feature importance chart

We will now comment on Figure 2, which applies the "Feature Importance" method to help you see the best columns for recognising the importance of messages. Here we notice the presence of many embeddings, which are the basis of our research. We also note that the most important elements in our network are: sentence size, sentence sentiment analysis, 'rank_appearance' (place of the least frequent word in the sentence) and sentence importance score.

As far as the combinations are concerned, we initially tried to keep everything, then to keep only the most significant features. The changes in the results were minimal. Secondly, as we were using the LSTM module, we decided to keep only those features that made sense in a sequence of texts: sentiment and links with the previous sentences.

Model Choice, Tuning and Comparison.

In the quest, we examined both Machine Learning (ML) and Deep Learning (DL) methodologies. Our objective was clear: determine which algorithms could most effectively discern the critical from the commonplace.

Machine Learning Methodologies

Logistic Regression (Score: 0.55):

Logistic Regression was our baseline model. Its propensity for fast execution and interpretability makes it a go-to for initial feature impact analysis. However, its linear nature means it often oversimplifies the complexities of natural language, which is rich with idiomatic expressions and contextual nuances. Indeed, it does not take into account the links between models and try to linearly separate each feature that are probably not linearly separable. Its score of 0.55 reflected these limitations, as it could not capture the intricate decision boundaries required for our classification task.

We quickly noticed that the minority class (important messages) was under-represented (around 18%) in the data we had. This could be seen in the predictions, and in particular in the logistic regression. In fact, the threshold we usually use (0.5) for binary classification was not the most appropriate here. In fact, the data often yielded too few important labels and we regularly had to lower this importance threshold to values of up to 0.3 to boost the number of 1 labels.

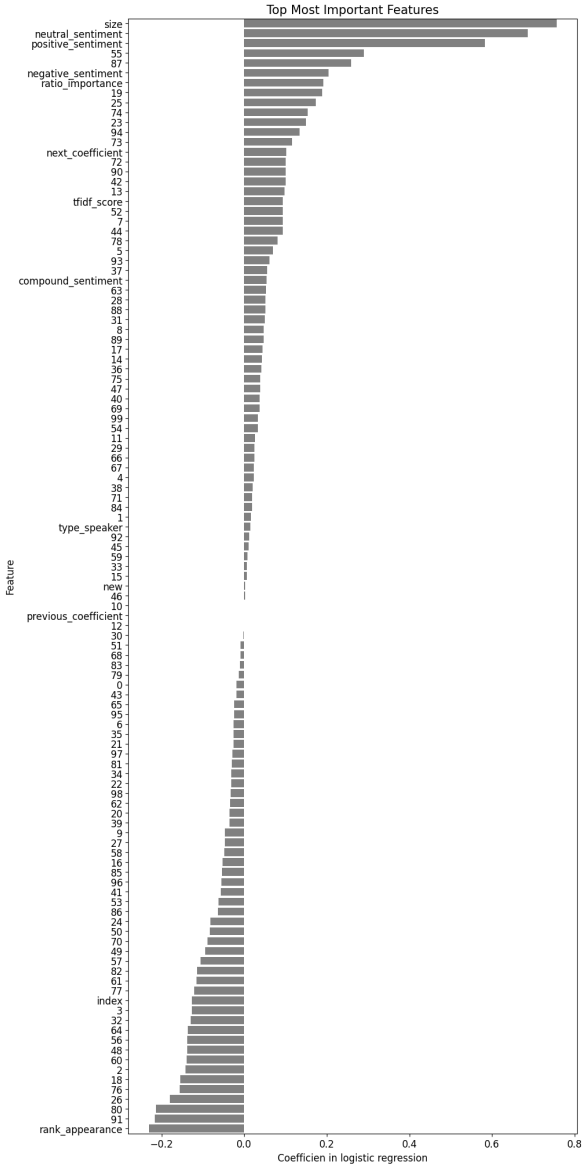


FIG. 2. Feature Influence Analysis (integers from 1 to 100 refers to the embedding using Word2Vec)

Support Vector Machine (SVM) (Score: 0.58):

SVM was chosen for its effectiveness. SVM is really robust against overfitting to some extent, which is essential when working with intricate datasets like the one we have. This robustness was reflected in its improved score of 0.56 (with balanced class and gaussian kernel). In our case, we used balanced SVM, which is useful when

one class (in our case, the class is labels that are 0) is much more frequent than another. To reduce bias and improve performance, balanced SVM weights the classes and maximises the margin between the two classes.

Decision Trees and Ensemble Methods (Random Forest Classifier (RFC) and XGBoost) (Score: 0.59):

As we have seen during TD classes, decision trees offer a more granular approach to classification, carving the feature space into decision nodes and leaves. This could be important when tackling the data we have. However, they are prone to overfitting. This is why ensemble methods like Random Forest Classifier and XGBoost, which combine multiple decision trees through bagging and boosting respectively, are valuable but still sensitive to depth parameter. Both methods scored 0.56 with max depth between 5 and 7.

Combination of different classifiers (Score: 0.58):

Once we had obtained several predictions from different models, we had the idea of aggregating them to try and improve our predictions as much as possible. On this subject, we tried to see the quality of the predictions of our different models. In fact, we realised that there are on average 18% of important messages, whereas our predictions using the previous models only suggest around 10% of important messages. This is why, instead of averaging the different predictions, we came up with the idea of assigning the label 1 as soon as a message was considered to be important among our different classes. Unfortunately, this apparently promising method did not improve our score or the quality of our predictions.

Deep Learning Methodologies

Artificial Neural Networks (ANN) and LSTM (Score: 0.59):

In our Neural Network, we tried multiple ideas. On the one hand, we implemented a 'classic' neural network as we saw during courses: this implied using a three-layered neural network, with two linear layers and an output layer. We noticed an important aspect of this Neural Network while making it: having an increasing number of nodes per layer helped us having better results. To this simple first approximation, we tried multiple complexifications. First, we implemented an LSTM Classifier. This is a recurrent Neural Network that can have feedback connections, which means that it can process entire sequences of data adn their links, whereas our previous model only processed a single data point. Our score was indeed better in this case (up to 0.59), as LSTM's sequential nature allows the Neural Network to understand the context and nuances in the text. Indeed, we gave him a sequence of 7 messages, their previous links and the sentiment analyser score. Thus it was able to imporve considerably the prediction taking into account the previous and following sentences. Secondly, we also tried to add a fine-tuning process. This means that we have tried to train BERT's embeddings while training the Neural Network simultaneously. BERT is initially pretrained, then we integrated it as the first layer of our Neural Network. The output of the BERT layers becomes the input of the subsequent layers. After, the error is backpropagated through the entire network, updating both the weights of BERT's model, as well as our initial neural network. Specifically, we used smaller learning rate for BERT's layer as it was initially pretrained. The first issue in this model was overfitting: as BERT has a tremendous amount of parameters, we overfitted in only 3 epochs our model. The second issue was the time to process: this Neural Network needed to much time to work, even though the results were actually good.

To prevent overfitting in this case, we added each time a validation set, which was alternatively 0.03 to 0.1 portion of the total data, and we calculated the loss each epoch, so that we could see overfitting when our validation loss begins to increase each time the epoch increases.

Graph Neural Networks (GNN):

Given interconnected nature of dialogue, the structure of Graph Neural Networks seemed to be really promising. Indeed, we could then be able to treat messages and their relationships as nodes and edges (with different kinds) in a graph. Thanks to GNNs we learn the importance of a message not just based on its content but also its context within the dialogue flow. This approach is particularly suited for dialogues where the significance of a message is often determined by its position and connection to other messages. We implemented such a GNN by building a message graph for each conversation, with links between each message. We then passed this data on to a classic PyTorch GNN architecture. Unfortunately, although we managed to implement such a graph structure, the results were disappointing, in particular because the values in the output layer were not easily separable, even when we increased the number of epochs and the batch size.

Conclusion

Our work was divided into two main parts: shaping the data so that the dataframe was optimized for the best results possible, and computing the best model possible. On the first task, we thought that the most important was to choose wisely the embeddings, even though added features have an important impact. On the second part, we thought that the best option in this task was the LSTM Neural Network, as it was able to modelise really thin variations and context.

Appendix

Preliminary approach and statistics

Before embarking on any search for a model and algorithm that could help us implement the classification we were asked to carry out, we began by gaining an overview of the data available to us.

Proportion of important messages: Before going any further into the patterns we might recognise within messages and the links between them, we can first look at the proportion of important messages in our training data. The figure is 18%. It's important to remember because we'll see later that many of the models we've implemented tend to underestimate the number of important messages and therefore reduce the f1 score, which is our evaluation criterion.

Manual study of sequence of messages: We quickly tried to draw a graph of our transcriptions to try and understand the criteria on which the labels assigned to the messages were based. After spending a great deal of time

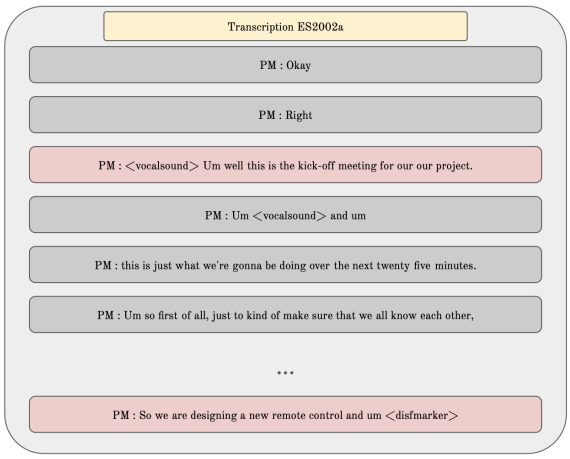


FIG. 3. 20 first messages taken from conversation "ES2002a" (red refers to important ones)

analysing and comparing important and non-important messages, we were unfortunately unable to identify any patterns that would allow us to separate the 2 classes of messages in one go.

Message length: At the same time, we tried to take a closer look at message length distributions, to see to what extent a long message would be more likely to be labelled as important.

As can be seen, messages labelled as important are generally longer than those that are short. In terms of the number of characters per message, we have an average of 29 for non-important messages compared with an average of 69 for important messages. We can then assume that the size of the messages will potentially be a means of partially separating the two classes, but this will certainly not be sufficient.

Type of speaker: Given that we have information about the "speaker" (PM, ME, ID, UI) in the transcripts, we

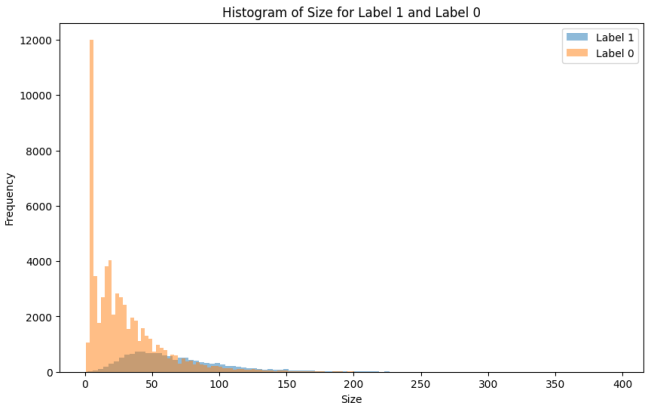


FIG. 4. Distribution of the size of the messages for each labels

can try to look at the proportion of each speaker and the proportions of important messages according to the different speakers (it is a priori conceivable that the Project Manager is more likely to write important messages than his colleagues). The proportions of messages sent by each speaker are as follows:

Speaker	Proportion
PM	33%
ME	23%
ID	23%
UI	21%

TABLE I. Breakdown of speakers

Speaker	Proportion
PM	18.2%
ME	18.1%
ID	18.2%
UI	18.8%

TABLE II. Proportion of important messages per speaker

Study the words among the important messages: Although we didn't directly notice any word patterns that could be used to distinguish between important and unimportant messages, we subsequently verified this intuition quantitatively by looking at the most represented words among the class of important and unimportant messages.

Study of links before and after messages: Given that we had at our disposal the links between messages (elaboration, explanation, ...) we tried to implement a method that takes into account the links with previous messages, and the links with subsequent messages (potentially multiple) in order to take them into account.

Even if we note that acknowledgments or corrections very rarely result in a label 1 for the message studied, we are unable to determine any real tendency for a message to be considered important in view of the link between it and the previous or following message. Perhaps we

Word	Number of Appearances
yeah	11986
uh	7588
okay	4719
well	2622
like	2604
thinks	2398
...	...

TABLE III. Number of appearances of top words in non-important messages

Word	Number of Appearances
uh	4783
think	1393
like	1355
remote	1189
one	851
well	803
...	...

TABLE IV. Number of appearances of top words in important messages

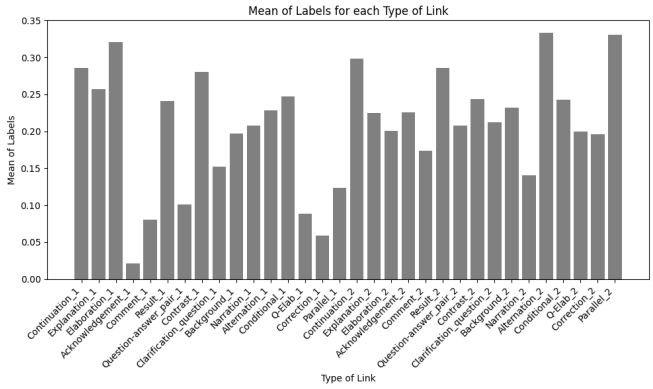


FIG. 5. Proportion of important messages per type of links: _1 refers to the link preceding, and _2 to the link following

need to look at higher degrees of relationship, or simply consider the links as additional information rather than an absolute. In any case, this lack of correlation between labels and types of links has led us not to consider the links at first and to concentrate on the content of the messages.

Study of embedding

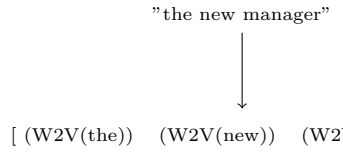
In order to take into account all the elements we had at our disposal, we quickly realised that there would need to be a message encoding part to enter the model, that there would potentially be some elements to be drawn from the secondary data such as the type of speakers, the semantic, orthographic and grammatical characteristics within the messages and finally, a part concerning the graphs with the links between messages from different speakers.

Sentence embedding using BERT-encoder

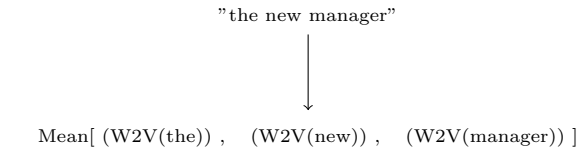
In our analytical progression, we initiated with embeddings—BERT initially offered a comprehensive context-aware vectorization of text/sentences, capturing nuanced semantic meanings from sequences of words rather than isolated terms (which is particularly relevant in our case). This deep learning model, pre-trained on a large corpus, excels in understanding the complex structure of sentences. However, its uniform performance across different types of machine learning tasks was not guaranteed.

Word embedding using Word2vec

Observing this, we incorporated Word2Vec, an alternative that processes words individually vectors representation given a model. The advantage of Word2Vec is that you can either take one pretrained library as word environment to transform into vectors, particularly Google's pre-trained vectors, or you can train you own model of embedding given a list of words. It is more accurate for customized word vectors that could potentially capture domain-specific language more accurately. The only issue regarding Word2Vec is that it is not a sentence embedding model, it only tranforms words to vectors, so given a message of undefined size, one cannot directly get one 1D vector for each message.



To overcome this problem, we have 2 options: the first is to transform our list of word-vectors into a single vector using an operation such as the maximum, the average or a weighted average (we'll see later how we tried to implement such an average weighted by the weight of the words according to TF-IDF).

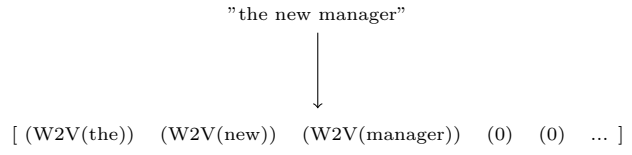


This method works, but reduces the importance that can be attached to word sequences. In fact, we understand that if we carry out the average, we will only retain partial information, which will turn out to be false when the sentence contains many unimportant words and a single important word which gives it the category of "important message". We will then be unable to recover this property by applying the mean. In the same way, we can understand that by applying the maximum (in terms of norm), this removes the meaning of the embedding because we only keep the largest vector (supposed to be the most important) and we lose any complementary information which would be encoded by another vector. In

particular, 2 different sentences containing the same important word (maximum argument of the sentence) will be encoded in the same way and we will be unable to distinguish between them...

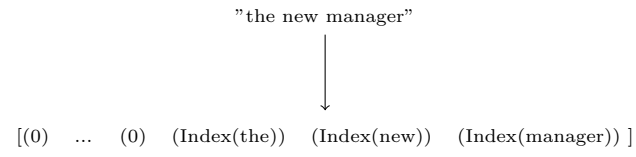
The other method that can be used to go from embedding words to embedding sentences is to keep our succession of vectors within a matrix whose columns are this succession of vectors. However, once again, we realise that there are going to be potential problems, as the matrix we have to create has to have the same size for all the messages we have. What can we do in this case? One idea is to determine a matrix size (number of words) that we can initialise to 0 everywhere and to fill in this matrix with the word-embeddings we obtain for each successive word. The final matrix may then be made up of many columns containing only 0s, which doesn't seem very coherent to us either.

Besides all, the best idea would have been this last matrix method. However, for memory reasons, we have decided to keep the "maximum" method.



Word embedding using `Texts to sequences`

The final way we have tried to embed sentences way using text to sequence method. Its purpose is to convert each word into a number that refers to its index. So you first need to create a set of vocabularies with an associated index, so that each word has an associated integer. Then, for each sentence, we transform each word into a list of integers corresponding to their respective indexes. Finally, we normalise all the vectorized sentences to give them a unique size so that they can be passed as input to future models.



Word embedding using TF-IDF

A final common option for vectorizing text is to use TF-IDF ("Term Frequency Inverse Document Frequency"), a calculation method used to determine the relevance of a sentence within a document and in relation to a group of documents.

TF-IDF Formulas:

- t: term we study,
- d: document where the term is take,
- D: corpus of documents.

$$TF(t,d) = \frac{\text{Number of times } t \text{ is in } d}{\text{Total number of terms in } d} \tag{1}$$

$$IDF(t,D) = \log\left(\frac{\text{Total number of documents in } D}{\text{Number of documents containing } t}\right) \tag{2}$$

$$TF\text{-}IDF(t,d,D) = TF(t,d) \times IDF(t,D) \tag{3}$$

The TF-IDF score obtained for each word is in fact an excellent measure of the importance of the word in a specific conversation, relative to the corpus as a whole.

Word embedding using `Embedding` from PyTorch

This feature of PyTorch is particularly useful when we have tried to implement embedding directly within a neural network. In addition, it allows embedding to be carried out directly within the neural network and optimised at the same time as the model is optimised.

1. Conclusion about embeddings

Throughout the project and during our various research projects, we tried to test these different embeedings on different models in an attempt to select the one that would work best on our data.