NLP Challenge: Summary Source Prediction

Paul Théron, Jérémie Dentan, Louis Gautier

Abstract—**In this data challenge, we investigate how to determine whether summaries of some news article were written by humans or machine generated. The issue was thus to extract some relevant features to capture the stylistic differences between these two kinds of summaries. Performing a quick data exploration allowed us to conclude that we needed two types of features to perform this task: features on the style of summaries and features comparing the style of summaries with the one of their associated documents. Leveraging pretrained language models such as BERT or GPT2, as well as statistics on the n-grams of the summaries and documents and metrics such as ROUGE scores, we managed to design such insightful features. We finally used these features to fit an XG-Boost classifier for which we optimized its hyper-parameters specifically for this task.**

This paper is provided with a git repository accessible **here**.

I. Introduction and challenges

Text summarization refers to the task of producing a shorter text to sum up the main ideas of a document. In the recent years, progress made in the field of transformer models such as BERT or GPT2 have led to the creation of algorithms capable of having human-comparable performances on this particular task [13]. As a result, determining whether a summary has been written by a human or machine-generated is becoming increasingly important.

This project consists of determining whether summaries of news articles were written by a human or generated with a model. It implies numerous challenges. First, as text summarization models are today capable of reproducing human-looking summaries, it entails to design features that reflect the style of summaries more than their syntactical correctness. Besides, we need to use features that compare summaries with the original documents to find out patterns on how humans and models generate summaries. Finally, using an adapted classification algorithm is the last challenge that we have addressed in this project.

This project is part of a kaggle competition. We reached 94% accuracy and 4th position on the leaderboard, which is accessible at: https://www.kaggle.com/c/inf582-2022/

II. Data exploration and pipeline

A. Data exploration

The dataset that we have at our disposal is composed of documents looking like news articles along with a corresponding summary, which can either be machine-generated or written by a human. It is divided into a perfectly balanced train set between these two classes containing 8,000 documents and a test set containing 3,200 documents.

When looking more closely at examples from this dataset, we can see that most of the machine generated summaries are almost perfectly syntactically correct except than some minor errors. For instance, the following summary from the train set, which was generated by a model, contains a repetition of ”and” and lacks a ”a”, which makes it likely to be machine-generated.

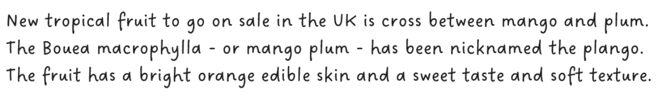
New tropical fruit to go on sale in the UK is cross between mango and plum. The Bouea macrophylla - or mango plum - has been nicknamed the plango. The fruit has a bright orange edible skin and a sweet taste and soft texture.

Fig. 1. Example of summary with some grammatical imperfections

However, some other summaries are perfectly syntactically correct and very hard to differenciate with human-written summaries, such as the following which was generated by a model:

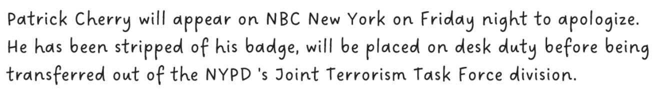
Patrick Cherry will appear on NBC New York on Friday night to apologize. He has been stripped of his badge, will be placed on desk duty before being transferred out of the NYPD 's Joint Terrorism Task Force division.

Fig. 2. Example of perfectly syntactically-correct summary

What this data exploration phase taught us was that these summaries had probably been generated using state-of-the-art language models. As a result, we will need to take that into consideration to design features that compare summaries with summaries which would have been written by such models, or to use such language models to generate document embeddings which would allow us to perform this classification task efficiently. Besides, as the discrimination between human-written and machine-generated summaries seems to be difficult, we will need to add features that compare summaries with the corresponding original documents.

This data exploration phase also taught us that the data has already been quite well cleaned and preprocessed. Words are properly tokenized and separated by spaces in the texts, while the train set is perfectly balanced between the two classes. In this project, we ran a supplementary preprocessing pipeline, which performs various tasks such as properly formatting numbers, special characters, and remove stopwords when needed.

Finally, we also have 50,000 news articles at our disposal, similar to the original documents from the dataset. We did not use those additionnal articles, but a way of improvement for our pipeline would be to fine tune every pretrained language models on those documents.

B. Our pipeline

The general idea of our pipeline was to generate as many relevant features as possible, then fed them to a classifier. We can distinguish two types of features:

- Features generated with summaries and labels only. These features attempt to separate summaries based solely on

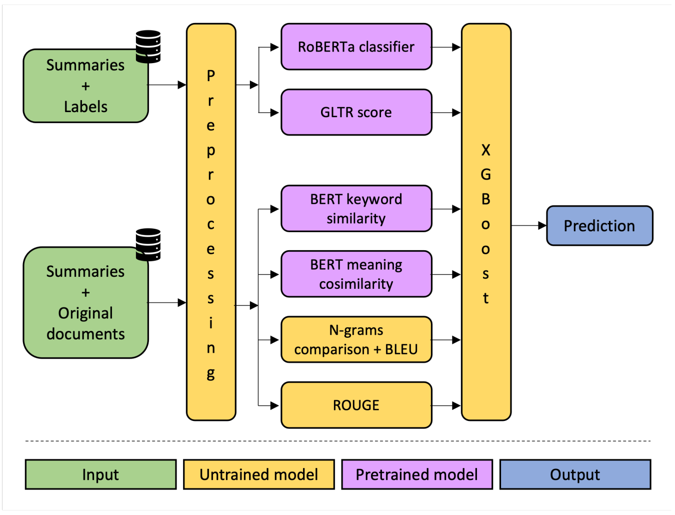

Fig. 3. General presentation of our pipeline

their writing style, without comparing them to the original documents. There are obtained on the one hand with a RoBERTa model finetuned for this classification task, and on the other hand with a score provided by a GLTR network.

- Features generated by comparing summaries and original documents. We computed four families of features: keyword comparison, cvosimilarity of the BERT meaning, n-grams comparison, and ROUGE score.

Eventually, we fed all those features to a classifier (here: XGBoost) as described in section V.

III. Feature building on summaries

As we mentioned in the introduction, the key challenge of this project was to design insightful features to feed a classifier with. First of all, we focused on building features taking only the summaries into consideration.

A. Doc2Vec embeddings attempt

In a first attempt, we tried to build embeddings for summaries and documents. Our underlying idea was to evaluate the distance between a document and its summary, hoping that this would provide a relevant feature for classification later on. To achieve this goal, we turned to the doc2vec technique. It is an unsupervised algorithm described in [1] that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents. The algorithm represents each document by a dense vector which is trained to predict words in the document. Indeed, this technique focuses on the document as a whole, to capture its deep meaning, to overcome the weaknesses of bag of words models. Unfortunately, this approach did not work and gave accuracies close to 50% (the random classifier). We think that it is due to the quality of the computer generated summaries, which are surely generated with high-level transformers. Therafter, the technique of a doc2vec trained from scratch is too simplistic for the data, and the embeddings produced do not show any major difference according to the type of summary.

B. Finetuning RoBERTa model for classification

Based on the findings of the previous model, we wanted to move towards more advanced techniques.

- Firstly, we wanted to look for a pre-trained model, which we would fine-tune with the training data. Indeed: the training set is very limited, and it is difficult to train large and efficient models with so little data.
- Then, we looked for a state of the art model, and chose RoBERTa, given the significant results described in [3]

As described in [2] BERT is at its core a transformer language model with a variable number of encoder layers and self-attention heads. The architecture is ”almost identical” to the original transformer implementation. BERT was pretrained on two tasks: language modelling (15% of tokens were masked and BERT was trained to predict them from context) and next sentence prediction (BERT was trained to predict if a chosen next sentence was probable or not given the first sentence). As a result of the training process, BERT learns contextual embeddings for words. After pretraining, which is computationally expensive, BERT can be finetuned with less resources on smaller datasets to optimize its performance on specific tasks. However [3] explain that BERT was significantly undertrained and propose an improved recipe for training BERT models. The new transformer has indeed the same architecture as a classical BERT, but modifications include:

- Training the model longer, with bigger batches, over more data
- Removing the next sentence prediction objective
- Training on longer sequences
- Dynamically changing the masking pattern applied to the training data

The significant changes made during the training phase allowed a significant increase in BERT results for datasets such as the classic SQuAD (Stanford Question Answering Dataset), or GLUE (General Language Understanding Evaluation) tasks

In practice, we fine tuned the model *roberta-base* of the HuggingFace library, we added a last layer of binary classification on the pre-trained core and we fine-tuned the model using the only tokenized summaries. With this first implementation, **we obtained a preliminary result of 0.89 taking into account only the summaries**.

C. GLTR: detecting similarities between summaries and texts generated by large language models

Language models define probability distributions over the tokens of a language with a given context as input. As a result, they are particularly useful for text generation. In the recent years, neural language models, which compute a vector representation for the previous context, and a probability distribution for the next token thanks to neural networks, have significantly increased in performance. Most of them are based on transformers, which refers to auto-encoder architectures using the principle of self-attention. This mechanism allows these models to evaluate the relationships between the words in the context and to better capture the semantic structure of sentences. Most advanced language models include Bert [2], or GPT2 [4].

As we saw in the data exploration part, it is likely that the summaries of our dataset were generated by a large language model such as BERT or GPT2. As a result, we had the idea to build features that reflect the level of similarity between summaries and texts generated by large language models. The statistical model called Giant Language model Test Room (GLTR) [5], is well adapted to this task. For each word of a text, given the previous words as context and a certain language model, this model computes three metrics:

- The probability of this word according to the language model,
- The absolute rank of the word among the possible words predicted by the language model,
- The entropy of the predicted distribution of words, which assess to what extent the context allows the language model to be confident about its next prediction.

These metrics are meant to give reliable indication on whether a text has been generated by a large language model or written by a human, which is perfectly adapted to our summary source prediction task. Texts written by human will thus typically contain tokens that have high absolute ranks and lower probability than machine-generated texts.

In the scope of this project, we used GLTR with a GPT2 model that we fine-tuned for text generation on the dataset of 50.000 documents provided in this data challenge, based on this article [6]. Then, we used the original code of GLTR to compute these metrics on each token of our summaries and we produced several features for each summary:

- The mean and standard deviation of the word probabilities and absolute ranks
- The maximum absolute rank of a token in the summary
- Fixed histograms reflecting the distribution of absolute ranks of tokens, probabilities and entropy in the text.

As we will see in part V-C, adding these features to our classifier allowed us to improve the accuracy of our classifier by approximately 2%.

We also tried to apply 1D convolutions of various sizes at the output of the vector of probabilities and absolute ranks, so that it could detect patterns that correspond to generated texts. We used batch normalization and dropout between each convolutional layer to regularize the model and a fully connected layer afterwards to classify the summary as human-written or machine-generated. Unfortunately, this approach didn't work and gave accuracies close to 50% (the random classifier). We think that this is due to the fact that the number of summaries in the training set was too small to be able to train a convolutional network from scratch.

IV. Feature building for comparison between documents and summaries

The features generated in the previous section were quite effective and allowed us to achieve a decent score (about 89% accuracy). However, they only focus on the abstracts and their writing style to perform the classification, and thus do not exploit the original documents. We then set out to compare the abstracts and the original texts on several aspects, and to add these comparison scores to our classifier.

A. *Keywords comparison with BERT*

The first comparison score concerns the keywords of the document and its summary. The idea of this approach is to evaluate the quality of the summary produced: if it is of good quality, the keywords of the original document should be found in the summary, and vice versa. We have therefore calculated the following four features :

	doc	**docKw**	**sum**	**sumKw**
% of document keywords in...	-	-	✓	✓
% of summary keywords in...	✓	✓	-	-

TABLE I
The features we computed to compare the keywords in the original document (docKw) and the summary (sumKw)

There exist many techniques to retrieve keywords from a given document, however we used the minimal technique presented by [7]. The process is the following :

- Removing stop words with nltk.
- Computing the mean BERT embedding of the document.
- For each non-stop-word, compute its BERT embedding and the cosine similarity with the BERT embedding of the whole document.
- Select the most similar word.

Adding those features did not improve the accuracy on the test set, probably because our keyword retrieval pipeline is really simple. Using state-of-the art keyword extractor would be an interesting way of improvement for our models.

B. *Meaning similarity with BERT*

Then, we wanted to compare the meaning of the original document and their corresponding summary. An estimation of this meaning can be obtained with advanced frameworks such as BERT or RoBERTa. To compute our score, we computed the mean embedding of the document and the corresponding summary, and then computed the cosine similarity between then. We did it with two different networks : BERT (with huggingface's "distilbert-base-nli-mean-tokens") and RoBERTa (with huggingface's "roberta-base-nli-mean-tokens"), so we obtained two features.

Those two additional features improved our score **from 0.90687 to 0.91000.**

C. *N-gram comparison and BLEU score*

Then, we added n-gram comparison features. The idea of those features is that summaries generated with frameworks such as GPT-2 or BERT are more likely to reuse the n-grams than human-written summaries. Thus, we count the proportion of alphanumeric n-grams (i.e. without punctuation) of the document reused in the summaries for $n \in \{1, 2, 3, 4\}$. This provides us four features derivated from the n-grams.

In addition, we added a BLEU score between the original documents and their corresponding summaries. This score has been introduced by [8]. However, we used a simplified version of this score (where used the same weight for all n-grams, which corresponds to a geometric mean) :

$$BLEU = \exp\left(\frac{1}{4}\sum_{i=1}^{4}\log(p_i)\right)$$

Where p_i is the proportion of reused i-grams from the document in the summary.

Those five features enabled us to significantly improve our accuracy score on the test set : **from 0.91000 to 0.93312.**

D. ROUGE scores

Then, we computed the ROUGE score of the similarity between a document and it corresponding summary. This score has been introduced by [9] and is also based on n-gram comparisons. We used the implementation of ROUGE score provided by [10]. For a given size of n-grams n, there are three ROUGE score which are computed as follows :

$$ROUGE_n = \begin{cases} ROUGE_{R,n} = \frac{|N_{sum} \cap N_{doc}|}{|N_{sum}|} \\ ROUGE_{P,n} = \frac{|N_{sum} \cap N_{doc}|}{|N_{doc}|} \\ ROUGE_{F,n} = \frac{2}{ROUGE_{R,n}^{-1} + ROUGE_{P,n}^{-1}} \end{cases}$$

With N_{sum} = n-grams of the summary and N_{doc} = n-grams of the document. In a way, this ROUGE-score is a precision, recall and f1-score over the n-grams of the document. Thus, $ROUGE_{P,n}$ is nothing else than the BLEU score we just implemented, and ROUGE is a generalization of BLEU. We computed the ROUGE score for $n \in \{1, 2, 3, 4, 5\}$.

The ROUGE score features significantly improved our accuracy on the test set, **from 0.93312 to 0.94067**.

V. Classification

Once our features were designed, the last part of our pipeline consisted in finding and fine-tuning a strong classifier to allow us to reach a maximum classification accuracy based on the features we had at our disposal.

A. Algorithm comparison

We first compared several algorithms with the default hyperparameters provided by the most used libraries in Python. The results are presented in table II.

We used logistic regression as a baseline, along with ensemble methods such as Random Forest and XG-Boost. The main difference between these two algorithms is that Random Forest builds several trees independently and combines them at the end of the process with bagging [11] while XG-Boost builds one tree at a time, with the new trees aiming at reducing the error committed by previous ones thanks to gradient boosting [12].

Accuracy on...	**Logit model**	**Random Forest**	**XG-Boost**
Train set	0.951125	1.00000	0.97425
Test set (Kaggle)	0.92500	0.93812	0.93937

TABLE II
Performance given by several classification algorithms with default choice of hyper-parameters and all our features

B. XG-Boost hyper-parameter tuning

Once we had determined that XG-Boost was the method that yielded the best results, we decided to fine-tune its hyper-parameters using the *sklearn GridSearch* function, which compares several choices of hyper-parameters while training the associated models with k-fold cross-validation.

We decided to tune several parameters:

- *n_estimators*, which corresponds to the number of base learners (base trees) used in this Boosting algorithm. We tried several values between 50 and 1000 for this parameter, and the optimal value turned out to be relatively small: 100. This is probably due to the fact that the model quickly reaches a saturation in accuracy as this parameter increases. As with this algorithm, new trees are sequentially added to correct the errors made by the previous trees, after a certain point, it becomes increasingly difficult for trees to correct the approximation error which tends to 0.
- *max_depth*, which corresponds to the size of these decision trees. We tested several values between 2 and 10, and the one that gave optimal results was 3. It is probably linked to the fact that deep trees tend to overfit the data, especially with the relatively small number of points in our dataset, while too small trees don't manage to sufficiently capture the significant patterns in the dataset.
- *learning_rate*, which refers to the weighting of the new trees which are sequentially added to the model. We tested several values between 0.001 and 0.1 and the one that gave the best results was 0.01, which seems to be a good compromise between sufficient convergence speed to avoid local minima and sufficient granularity to correctly explore minima.
- *colsample_bytree*, which refers to the fraction of the columns randomly uniformly chosen for building each tree. We obtained an optimal value of 0.7, which allows the base learners to take enough elements into consideration, while still preventing overfitting with this random sampling which creates an artificial "noise".

C. Evaluation of features performance

It is finally possible to build a tree of importance of the features in the classification thanks to XGBoost. Indeed, during training, the classification algorithm uses a hierarchy of features to give more importance to those that are the most decisive. As expected, RoBERTa has a very strong importance in the classification, which was shown by the gain in accuracy when we integrated it. However, the result highlights the importance of n-grams for classification. Indeed, one can suspect that automatic summary generation algorithms tend to take grams of high importance identically in the original document for the summary; whereas a human will not consciously identify them and therefore will almost certainly reformulate the sentence.

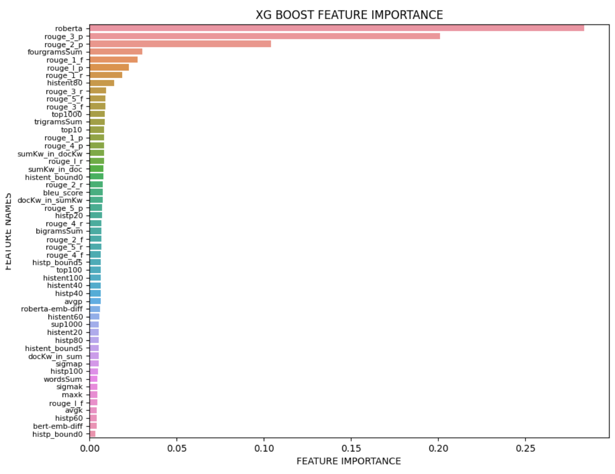

Fig. 4. Highlighting of feature importance

	RoBERTa	GLTR	Keywords	N-grams	ROUGE
Score	0.89	0.90687	0.9100	0.93312	0.94067

TABLE III
SUMMARY OF THE EVOLUTION OF RESULTS WITH FEATURES

VI. Concluding thoughts

A. *Ideas for improvement*

We identified two main ideas for improvement:

- Some features could be improved. In particular, the keyword comparison could be made with state-of-the art key word extraction framework, which should improve the relevance of this feature for the classification.
- Future work on the project could be interested in deepening the training of RoBERTa transform, with the dataset of news documents, on the same basis than described in [3] to finally take up the pipeline we proposed.

B. *Conclusion*

In order to determine weather summaries of news article were written by humans or machine generated, we leveraged state of the art models such as BERT and GPT2, to be efficient in false summary detection, in addition, we computed metrics such as n-grams occurrences between documents and summaries that we combined with transformers features with an XGBoost classifier. With this model, **we reached 94% accuracy and a first place in the kaggle leaderbord**.

References

[1] Quoc V. Le, Tomas Mikolov : Distributed Representations of Sentences and Documents, 2014

[2] Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina Bert: Pre-training of deep bidirectional transformers for language understanding, 2018

[3] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov: RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019

[4] Radford, Alec and Wu, Jeffrey and Child, Rewon and Luan, David and Amodei, Dario and Sutskever, Ilya and others Language models are unsupervised multitask learners, OpenAI blog, 2019

[5] Gehrmann, Sebastian and Strobelt, Hendrik and Rush, Alexander M GLTR: Statistical detection and visualization of generated text, 2019

[6] How to Fine-Tune GPT-2 for Text Generation, TowardsDataScience https://towardsdatascience.com/how-to-fine-tune-gpt-2-for-text-generation-ae2ea53bc272.

[7] Maarten Grootendorst, *Keyword Extraction with BERT*, https://towardsdatascience.com/keyword-extraction-with-bert-724efca412ea and it associated github repository https://github.com/MaartenGr/KeyBERT

[8] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. *Bleu: a Method for Automatic Evaluation of Machine Translation*. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

[9] Chin-Yew Lin. 2004. *ROUGE: A Package for Automatic Evaluation of Summaries*. In Text Summarization Branches Out, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

[10] Paul Tardy, ”rouge” git repository https://github.com/pltrdy/rouge

[11] Breiman, Leo Random forests, Machine learning, 2001

[12] Chen, Tianqi and Guestrin, Carlos Xgboost: A scalable tree boosting system, Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, 2016

[13] Liu, Yang and Lapata, Mirella Text summarization with pretrained encoders, 2019