Data Challenge - Kaggle - Summary Source Prediction

Guillaume BARTHE, Yi ZHANG, Ariel RAMOS

March 20, 2022

***Abstract*—Abstractive text summarization is the task of generating a short summary that captures the main ideas of the source text. Here the objective of this challenge is to study and apply machine learning & artificial intelligence techniques in order to be able to predict whether a summary is written by human or a machine. The overwhelming expansion of Deep Learning these days have led to very efficient text generation algorithms. Those models are able to produce short texts that sometimes a human is unable to differentiate from a human written version.**

Keywords - Summary, Artificial Intelligence, Neural Network, Embedding, Transformers, Binary text classification.

I. Introduction

A. Overview

The objective of this report is to summarize our approach to this data challenge which is hosted on **Kaggle** and you can find our code with this github link [1]. We are dealing with a binary classification problem using text data. The dataset represents documents written by humans and their summaries. Each summary is labeled positively (1) if it was written by a human or negatively (0) if it was generated by a machi,e. We will present how we processed the data, how we extracted features from the data and finally the architecture of our best model.

B. Evaluation Metric

The performances of our models will be assessed using the **Accuracy metric**. This metric is defined as the ratio between the number of correct predictions divided by the total number of predictions. Therefore the objective of our work was to minimize this quantity in order to get the most accurate model.

II. Data Analysis

A. Dataset description

We had access to a training dataset containing 8000 documents and their summaries. The machine generated texts were generated using the seq2seq transformer. This training dataset was equally divided between human summaries (4.000) and machine ones (4.000).

In addition to this training set we had access to a test set of 3200 documents and their summaries. This test set is not labeled and the objective is to use our trained model to predict the labels of the test summaries and then upload our predictions on Kaggle.

Finally, we have acces to a large set (50.000) of original documents written by human in case we want to create more training data.

[1] https://github.com/arielramos97/NLP-Challenge

B. Data preprocessing

Before using the texts as inputs of our model, we decided to clean the summaries in order to remove special characters for example that would make embeddings less accurate. We preprocessed the texts in the following way :

- **Step 1 :** Lowercase each summary to standardize the data
- **Step 2 :** Remove special characters and punctuation.
- **Step 3 :** Strip multiple white spaces.
- **Step 4 :** Remove stopwords with the help of NLTK library and apply stemming.

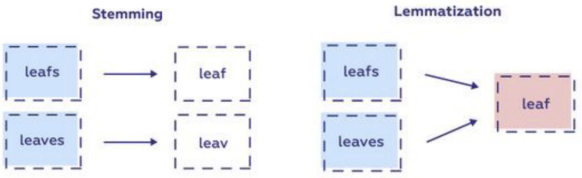

Fig. 1. Illustration of stemming and lemmatization

By doing this preprocessing we ensure that we have a robust training dataset and reduce the noise in the text embeddings.

III. Model Tuning and Comparison

A. Baseline - TF-IDF

Before exploring complex models we looked at the baseline given for the challenge and tried to improve it. The baseline relies on **TF-IDF vectorizer** [7] which is a very common word embedding that stands for term frequency - inverse document frequency. This is the product of the frequency of the term (TF) and its inverse frequency in documents (IDF). This method is usually used to measure the importance of a term i in a document j relative to the rest of the corpus. The idea here is to apply TF-IDF to the summaries and feed the vectors to a logistic regression model in order to predict the label of the summary. In the following table you can observe the accuracy results that we were able to obtain by using this model on either the raw or cleaned dataset.

Dataset used	Training time(s)	Accuracy
Raw Dataset	2	63%
Cleaned Dataset(Stopwords)	5	66%
Cleaned (Stopwords + Stemming)	10	69%

TABLE I
PERFORMANCES OF THE LOGISTIC REGRESSION MODEL ON DIFFERENT DATASETS (RAW OR PREPROCESSED ONE).

As you can observe, the improvement of the model is not really significant. Therefore, we decided to investigate more

complex models as well as trying to utilize both the original documents and the summaries.

B. Pretrained models - Concatenate model

Our first idea was to explore the training dataset and try to extract information from the summaries and the original documents. Indeed, it might be possible to find some features that would differentiate a machine generated text and we could use those features as input in a new model.

The features we thought would be interesting to look at were :

- **Average sentence length** : The intuition is that machine generated summaries might have a common sentence size.
- **Summary's length** : Similar to the previous one, generated summaries might have a common size whereas human ones are random.
- **Number of common words** : This is a sort of plagiarism detection. It might happen that the algorithm uses a lot of words already used in the document. This feature will therefore represent the percentage of words in the summaries that already appear in the original document
- **Longest common sentence** : Similar to the previous one, we look at the zie of the largest sentence that appear in both the summary and the document. The intuition is that most of the time a human won't repeat a whole sentence without changing any words.
- **Number of proper nouns** : A machine might capture in the document that some proper nouns are important and will tend to repeat all of them, therefore we scan for the total number of proper nouns in the summary.
- **Cosine similarity** : A similarity measure computed between the summary and the original document in order to measure how different is the summary. The intuition is that a human will tend to use different words that have the same context but don't appear in the original document

Now that we defined several features to extract from the data, the main question was how do we create a model that is able to deal with both text and numerical features as input ?

The idea we came up with was to concatenate two neural networks, one designed for the embeddings and training on the text and another one that trains on the numerical data.

For the numerical part we created a simple neural network with one fully connected layer which uses "ReLU" activation function. (We have tried adding more layers but this was slowing the training too much and not improving the model as much as we would have liked it to). We only kept 4 features among the 6 features mentioned above because they yielded more interesting results. For the textual data, our model uses a RNN (GRU [2]) after the embedding of the texts, and its output goes into a simple fully connected layer with activation function "ReLU". Then we use a concatenate layer to combine the output vectors from these two networks. We connect the concatenate layer with a fully connected layer which uses "sigmoid" activation function.

To make the concatenate model have optimal performance, we perform hyper-parameter tuning using the library "Keras Tuner" [5]. With this library, we decided to set the learning rate = 1e-4, hidden size of GRU = 128, batch size = 128 and dropout probability = 0.1.

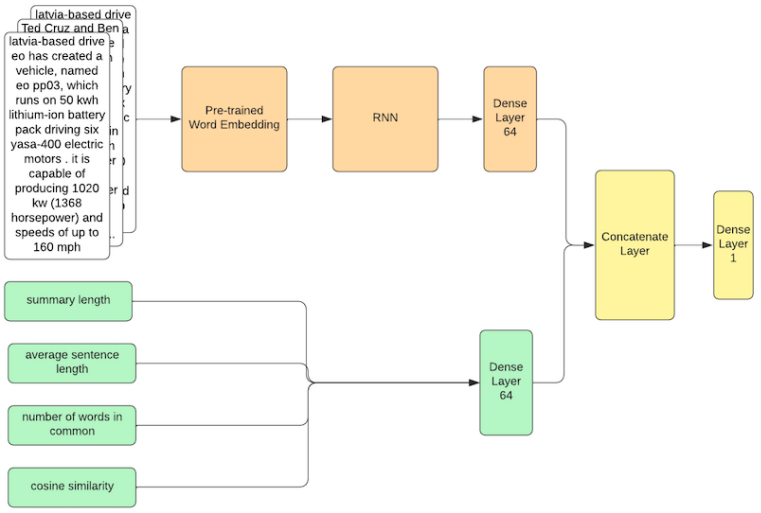

Fig. 2. Visualization of the model used for predictions. It combines the output weight vector of two different neural networks applied on textual data and numerical data

To compute the embedding of the summaries we decided to compare TF-IDF with the following pretrained embeddings:

- **Universal Sentence Encoder** : The USE [1] encodes text into high dimensional vectors that are commonly used for text classification, semantic similarity, clustering or other natural language tasks. The pre-trained Universal Sentence Encoder is publicly available in Tensorflow-hub.The intuition here is that this encoder summarizes any given sentence to a n-dimensional sentence embedding. Since the same embedding has to work on multiple generic tasks, it is able to capture only the most informative features and discard noise. Therefore, this will result in a generic embedding that generalizes to wide variety of NLP tasks, this is why we decided to try it on our data.
- **GloVe** : GloVe [6] stands for Global Vectors for word representation. It is an unsupervised learning algorithm developed by researchers at Stanford University aiming to generate word embeddings by aggregating global word co-occurrence matrices from a given corpus. Each word in the summary will be encoded into a 300-dimensional vector.

C. Transformers - RoBerta

Transformer models are a class of deep neural network models. They managed to improve the state-of-the-art scores on a wide variety of standard NLP tasks. **Huggingface library** [8] provides pre-trained models for a variety of transformer architectures and we decided to try some of those for our

classfication task. Indeed, these models have shown promise in improving results on tasks with a small amount of labeled data.

Historically, language models could only read text sequentially meaning either left-to-right or right-to-left but couldn't do both at the same time. BERT [3] was a revolution when it came out because it is designed to read in both directions at once. This capability, enabled by the introduction of Transformers, is known as bidirectionality. Because transformers can process data in any order, they enable training on larger amounts of data than ever was possible before their existence. This is the reason why a lot of pre-trained models like BERT, which was trained on massive amounts of language data, emerged after 2018.

BERT was pre-trained using only an unlabeled, plain text corpus (the whole English Wikipedia and Brown Corpus). The main difference with transformers is that they process any given word in relation to all other words in tje sentence, rather than processing them one at a time. Indeed, they look at all surrounding words and understand the full context of the work. This is a great improvement compared the traditional method of word embedding, in which previous models like GloVe would map every single word to a vector which only represent a small part of that word's meaning.

BERT uses a method of masked language modeling to keep the word in focus from "seeing itself". BERT is then forced to identify the masked word based on context alone. Therefore, words are defined by their surroundings and not by a pre-fixed value.

Nevertheless, scientists found that BERT is a significantly undertrained model and can be improved. RoBERTa (Robustly Optimized BERT Pretraining Approach) was therefore introduced. It uses much more training data (160Gb VS 16Gb), a dynamic masking pattern instead of static masking pattern and also it was trained on longer sequences.

IV. Experiments and Results

A. Model architecture

The model we used here is the **Roberta transformer** [4]. Similar to a traditional classifier it fits the summaries to the labels sent as inputs. The transformer first performs the encoding of the corpus based on its tokenizer and then realizes a forward pass on the neural network followed by an optimization step on cross entropy loss.

Indeed, the binary classifier is built on top of RoBERTa transformer. Hence, it corresponds to the base of the architecture utilized which can be summarized as:

- Roberta Model from pretrained 'roberta-base'
- Dropout layer with rate 0.3
- Linear layer from 768 to 64
- Normalization layer
- Dropout layer with rate 0.3
- Linear layer from 64 to 1
- Sigmoid layer

The inputs for the Roberta Model are the following:

- The input ids from the summaries and the documents which were computed by the Roberta tokenizer (also from 'roberta base').
- The attention mask list that indicates which tokens should be attended to, and which should not. This is useful when the summay text is shorter than 512 tokens.

B. Hyperparameters

In order to reach better performances we opted for parameter tuning. We tried several combinations and compared the results :

- Linear transformations: (768 to 64 and 64 to 1), (768 to 128 and 128 to 1), (768 to 256 and 256 to 1).
- Learning rate: 1e-05, 1e-06, 1e-07.
- Dropout rate: 0.1, 0.2, 0.3

The best performance was 94 % and it was achieved by employing the following combination of hyperparameters: linear transformations (768 to 64 and 64 to 1), learning rate 1e-05 and dropout rate 0.3.

C. Results

On the following table, you can observe some of the results that we were able to achieve with the different models that were presented in the previous sections. Please note that we used Google Colab Pro with GPUs otherwise some models (like Roberta) took 12 hours of training for 3 epochs. Furthermore, we noticed that, when playing with batchsize parameter, it was easy to reach the memory limit in colab and we were not able to test large values of batch size (Higher or equal than 32) for the Roberta model. Finally, the training time was measured on a 4 epochs training. This is an arbitrary number that we chose in order to reduce overfitting but this parameter can easily be modified when using a different dataset.

Model used	Batch size	Accuracy	Training time(min)
Logistic Regression	NaN	63%	0.03
Random Forest	NaN	62%	0.2
Concat Model (Glove + GRU)	128	80%	4
Concat Model (USE + Dense)	128	70%	11
Concat Model (USE + Dense)	256	72%	15.3
TFRobertaModel (Keras)	4	87%	29
TFRobertaModel (Keras)	8	89%	35
RobertaModel (Pytorch Model)	**8**	**94** %	**18**

TABLE II
PERFORMANCES OF MULTIPLE MODELS REGARDING THE BATCH SIZE AND EMBEDDING. CONCAT MODEL REFERS TO THE CONCATENATION OF TWO NEURAL NETWORKS PRESENTED ABOVE

From the table above, it can be observed the two versions of RoBERTa were mentioned: TFRobertaModel and RobertaModel. In practice, they are the same but TFRoberta is also a subclass of tf.keras.Model so it can be used as a regular Keras Model while RobertaModel is a PyTorch torch.nn.Module subclass.

V. Conclusion and Future work

The task of classifying human generated texts from machine generated texts plays an important role to mitigate the misuse of text generative models that may be utilized, for instance, to produce fake news. Hence, in the presented project the RoBERTa architecture has been applied to create a model

capable to distinguish human generated summaries from machine generated summaries.

An accuracy of 94 % was achieved which outperforms the baseline architecture that reached an accuracy of 62 %. This was possible by hypertuning the parameters of the model. Indeed, most of the time was spent setting different parameters and running experiments. Therefore it can be concluded that Huggingface models are sensitive to hyperparameters.

Finally, one of the limitations of the current approach is that RoBERTa can only handle texts that contain up to 512 tokens. Therefore, there were some sequences that needed to be truncated in order to work which means that the results obtained could be further improved by making the model read all the information. This could be done by separating the summaries into chunks and feed them to the model. Then, combine the values retuned into a single vector (by averaging them) and pass it to the subsequent layers of the model. This will potentially lead to make a more informed decision based on all the summary instead of only the first 512 tokens.

For future work, it would be interesting to find a way to use the file containing 50.000 original documents because in this study we did not manage to use it efficiently. Although we tried to use it to augment the size of our training data but it was creating imbalances between human made summaries and machine ones so we decided to discard this idea.

REFERENCES

[1] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018.

[2] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. cite arxiv:1907.11692.

[5] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Keras Tuner. https://github.com/keras-team/keras-tuner, 2019.

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[7] Juan Ramos. Using tf-idf to determine word relevance in document queries, 1999.

[8] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.