Create stories from song lyrics

Matteo Zaramella 2025806

Abstract

This project aims to create a new story starting from the information extracted from the lyrics songs. This is a new experiment in this field, because there are no models that now a days do this, at least in my knowledge, so it can be considered as a baseline. This scoe of this project is to amuse people with a simple story and try to expand the information extraction and text generation in a new field.

1 Introduction

This is the report of my project for the course "Elective in Artificial Intelligence: Narrative Understanding and Storytelling". I developed a pipeline to generate a story starting from the information extracted from the lyrics. This is a very challenging task because, to my knowledge, there are no datasets specified for lyrics information extraction and neither on generated stories based on lyrics available. For the information extraction task I did transfer learning using Named Entity Recognition from Spacey (NER) and KeyBert (KeyBert, 2020) which are trained on text and not on lyrics. For the grammar correction part, I used a T5 model (Raffel et al., 2020) finetuned on a grammar dataset, and for the text generation, a gt2 (gt2 OenAI) finetuned on the stories dataset.

All these models are merged in cascade and in parallel to obtain the best results possible.

2 System design

The process starts taking as input a lyrics path, downloading the lyrics, and passing it to KeyBert and NER model from Sacey. Respectively, KeyBert extracts the two more important and recurrent spans, composed of 1 to 5 words, instead, Spacey executes the Name Entity Recognition task returning the entities present in the lyric. Once obtained, all this information is merged substituting each pronoun or subject present in the two spans with an entity, from "person" or "organization", found by NER model, choosing first the most recurrent ones. After, the longest between the "data" entities found is added in front of each span, creating a phrase made from the song. The two phrases generated are passed first to T5, to correct eventual grammar errors, and then to gpt2 (finetuned on 80000 samples of Tiny stories section 4). Gpt2 generates two stories for each input phrase, both are evaluated with some selected metrics section 3, and the best one is selected based on ma In case no entities are found, the phrases are made using the two spans as they are. Then the phrases are first passed to T5 model to correct eventual grammar errors, and later, passed to gpt2, for text generation. Gpt2 returns two stories for each input phrase, and among these two, the best one is selected based on majority vote rules using some selected metrics section 3. In the end, we obtain the two best stories, one for each input phrase, and among them, we select the best one with the same metric mechanism. The metrics and statistics of the selected one are shown and it is returned as output. This output can be cut for a maximum of 30 characters if there is a finished phrase on them, otherwise, it is returned as it is.

3 Evaluation

Evaluating the text generated without any text reference as in this case, is the most difficult part. Nowadays, in my knowledge, there isn't yet a good metric that does it, because there are a lot of different aspects and variables to consider in a text. To evaluate the story generated I decide to use different indexes to obtain a final evaluation. More in detail I used many indexes of the library "Textstat" (library), which are not always reliable singularly, but using the majority vote rule, I obtained good results. The main indexes used are:

1. Flesch-Kincaid Grade Level (indices): It estimates the grade level required to understand

the text.

2. Gunning Fog Index (fog index): It measures the years of formal education required to understand the text.

3. Coleman-Liau Index (index, a): It calculates the grade level required to understand the text, considering the average number of characters per word and sentence.

4. Automated Readability Index (readability index): It measures the grade level required to understand the text based on the number of characters, words, and sentences.

5. SMOG Index (index, b): Simple Measure that estimates the years of education required to understand the text.

6. Flesch Reading Ease (indices): It calculates the readability of the text.

All the indices return higher values for text more difficult, except for the last one (Flesch Reading Ease) that return higher value for texts easier to read. I used one more index, the perplexity (Perplexity), it is calculated by comparing the probabilities assigned by the model to each word in the sequence with the actual words that occur in the test data. A low perplexity score for a text generation model indicates that the generated text is more similar to the training data and that the model is more confident and accurate in its generation. To use it without any text reference, I split my text into train and test, respectively 80 and 20 %, to obtain a similarity between them.

I calculated all these indices for both the two stories passed as input per time, and select the story with better indices value (considering every index vote as equal).

4 Finetuning

T5 fine tuning: At the beginning I used HappyTransformers (HappyTransformers) to finetune the T5 model for grammatical error correction. I used a part of "c4_200m" ($c4_200m\ byHuggingFace$), 100000 samples for the Training set and 20000 for the Validation set. The training was entirely done by the HappyTransformers library, without the possibility to manage it by me, so I preferred to train another model writing the code in Pytorch. I used 50000 samples from the "c4_200m" dataset, more in detail, 5000 for the Validation set and 45000 for the Training set. I used half the samples used with HappyTransformer due to the Google Colaboratory limitations, but still reached very good results. The training made in Pythorch was slower than the HappyTransformers library one, but, after only one epoch (around 1 hour and 40 minutes) the loss on the validation set passed from 2.863 to 0.083.

gt2 fine tuning: I have first finetuned the gpt2 (gt2 OenAI) model on the "FairytaleQA" dataset (by HuggingFace, a), removing all the questions and answers, and keeping only the story given as a reference. I utilized the stories as input but also as target data for finetuning the Large Language Model. The perplexity on the Test set was 36.34 before the training and after two epochs became 22, this improvement is not as much as expected, but the style of the text generated was different. I trained the model for two epochs because the dataset was quite small, only 2137 stories for the Training set, and training for three epochs caused overfitting. In a second phase, I finetuned the gpt2 on the TinyStories dataset (by HuggingFace, b), which is a bigger dataset with more samples. I trained two models one using a Training set of 40000 samples and another one with a Training set of 80000, and respectively 4000 and 8000 Validation and Test set samples, for one epoch. Finally, I tried to generate three stories based on three different input lyrics ("Perfect", "Flowers" and "Party in the usa") for each of the three models finetuned, and based on the output texts and metrics was clear that the models finetuned on the TinyStories dataset generated better stories. In the Table 2, is shown the perplexity obtained by the story generated by each model, on the three different songs, and is clear that the gpt2_80000 finetuned on 80000 Training set samples of Tiny stories obtained the best results. Based on this the model used on the pipeline is this one.

5 Results and conclusions

The results obtained are quite good, they improved a lot after the gpt2 and t5 finetuning. As shown in Table 1, the stories generated on the easier songs like "Flowers" and "Party in the usa" obtain a good Reading and ARI score, instead the stories generated starting from a rap song like "Without me", that is very complicated and full of different and long words, obtain a low Perplexity and Flesch-Kincaid score. On the other hand, the Table 3,

Song	Flesch-Kincaid	Fog	Coleman-Liau	ARI	SMOG	Reading	Perplexity
Empire state of mind	6	8.3	8.18	8.1	9.0	80.51	208.38
Perfect	4.5	5.94	6.13	5.0	8.3	84.57	181.25
Flowers	3.1	4.21	4.84	3.4	6.7	88.02	182.30
Party in the usa	2.1	4.15	5.25	3.7	6.8	96.08	169.80
Levitating	4.2	5.64	6.13	4.9	6.8	85.39	196.12
Without me	6.1	7.41	7.07	7.2	7.6	80.41	212.45

Table 1: Results of the best story generated for each song.

Song	gpt2_fairytales2eochs	gpt2_40000	gpt2_80000
Perfect	220.30	195.50	190.52
Flowers	214.35	182.54	185.19
Party in the usa	201.57	4.37	173.46

Table 2: Perlexity obtained by the finetuned gpt2.

Song	Unique words	avg length words (char)	avg length seq (char)
Empire state of mind	171	4.62	85
Perfect	172	4.33	58.35
Flowers	158	4.34	64.67
Party in the usa	151	4.37	46.44
Levitating	172	4.38	63.33
Without me	191	4.36	86.13

Table 3: Data about the stories generated conformation.

shows that despite the finetuning, the model still generates only small words, almost all of 4.30 average characters length. Another clear thing is that the generated words variability (unique words) depends a lot on the words presented in the song, for example, the story based on the information extracted from "without me", which is a song full of different words, has 191 unique words, way more of the other generated stories. In conclusion, the results obtained are not excellent, and there are margins to improve. The main difficulties of this pipeline are the information extraction and the text generation, both due to the lack of a specific dataset on which to train the models. More in detail, the information extraction is not always precise due to the different conformation of a lyrics song compared to a text, and this cause on cascade some errors and imprecisions in the text generation. To improve the results, I should create a hand-crafted dataset of information extracted from the lyrics and a stories dataset generated from the lyrics, this will improve a lot the performances.

References

Dataset FairytaleQA by HuggingFace. a. Dataset fairytaleqa.

Dataset TinyStories by HuggingFace. b. Dataset tinystories.

Dataset c4$_200m\ byHuggingFace.Datasetc4_200m.$

Gunning fog index. Wikipedia page for gunning fog index.

gt2 OenAI. gpt2 reference from openai.

HappyTransformers. Happytransformers package.

Coleman–Liau index. a. Wikipedia page for coleman–liau index.

SMOG index. b. Wikipedia page for smog index.

Flesch–Kincaid indices. Wikipedia page for flesch–kincaid indices.

KeyBert. 2020. Keybert github repo.

Textsat library. Textsat section of pypi.

Spacey NER. Spacey ner reference site.

Perplexity. Wikipedia page for perplexity.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer.

Automated readability index. Wikipedia page for automated readability index.