Judit Casademont Moner
03/11/2021

Machine Learning for Statistical NLP: Advanced LT2326

# T(AI)lor Swift, a Taylor Swift Lyrics Generator

## Final project report

### Introduction

The purpose of this project is to create a natural language generation (NLG) model that is capable of writing song lyrics in the style of the world-wide famous singer-songwriter Taylor Swift. By providing a few words from a song, the T(AI)lor Swift model generates the lyrics of a new song.

The main motivation behind this project is that, in the MLT programme, we have not yet worked with NLG nor tried to produce any kind of written output using machine learning that was intended to be similar to human language. Additionally, grammaticality and syntactic and semantic coherence come naturally to most human beings, however, machines require a lot of input data and computation processes in order to be able to autonomously "say" things in natural language. Therefore, I believe NLG is an interesting and relevant area of study and research. Finally, I was motivated by the question of 'can an AI imitate art?'; since art is subjective and extremely perceptional, I wonder to what extent a model created with machine learning methods would be capable of making art, and even achieve similarity to an artist's personal style.

The T(AI)lor Swift lyrics generator is structurally based on the 'Let's auto write the Deep Purple Lysics (Pytorch)' project published on Kaggle. From their idea, some methods were modified to fit the goals of this project, and some architectural changes decisions were made for the sake of experimenting and exploring the world of NLG.

This project is written entirely on a Jupyter Notebook, so to run the code, it is only necessary to run the Notebook cell by cell. To avoid retraining the model, do not run the model cell and run the torch.load cell instead, which loads the model saved as LSTaylorM.pt. Moreover, to be able to run the evaluation cells, having downloaded and installed Jury is needed; this can be done by running on the command line 'pip3 install jury' if you're working on your local machine, or 'pip3 install --user jury' when working on mltgpu. Once the installation is done, the evaluation cells can be run normally.

In this report, the following points will be briefly discussed: background, where we will see the dataset and other people's similar projects have worked with and achieved; methods, where the data processing, the design of the model, the lyrics generation and the evaluation metrics will be discussed; results, which will include a comment on what they represent and how they could be improved; and a short conclusion on the achievements and learning outcomes of this project.

Judit Casademont Moner
03/11/2021

**Background**

This project uses a half-handmade dataset. The base dataset is Kaggle's "Taylor Swift Song Lyrics from all albums", which was last updated in 2018. Since Taylor Swift has written three more albums until then, the data lyrics from these albums were collected, processed, and added to Kaggle's original dataset to create a more complete one. This way, more data to train the model was available. As we all know, when it comes to machine learning, usually having a decent quantity of data is important. This dataset contains 145 songs written by Taylor Swift, which add up to a total of 61,280 words, 3,348 unique words.

NLG is a task that has been attempted many times before to generate, describe or summarize text (both prose and poetry) in a manner that is as human-like as possible. This can be achieved by using different methods and strategies. Such methods could be divided into two main ones: those which use a model that was built from scratch, such as an RNN of some sort, and those which use a pretrained model, such as BERT or GPT-2. Natural Language Generation can be approached from a fill-in the gaps perspective, which is enclosed (usually within a fixed template) and allows for less freedom, or from a dynamic language generation perspective where, given some delimiting parameters, the AI outputs longer text in a more or less free manner. For this project, I decided to build an RNN, specifically an LSTM (Long short-term memory).

For the project, the source of inspiration to build the architecture was 'Let's auto write the Deep Purple Lysics (Pytorch)', a project published on Kaggle. What they do in that model is predict the next character given the 100 previous characters. In the project, I made the choice to predict the next word given the 10 previous words. My first intuition was that, by using words instead of characters, the model would perform better when producing natural language. However, this was not the case. This will be explained in the results and conclusions sections.

**Methods**

The dataset is stored in a csv file. Said csv file is read and stored in a Pandas dataframe to help visualize how the data is structured. Afterwards, each song, which in the dataset is found separated in verses, is appended into one line in order to be processed as one. This decision was made because I believe that, as the lyrics of each song merge together to produce one overall meaning, the most logical way is to treat the lines of each song as one group, and not as individuals. In addition, the decision was made not to remove punctuation, because punctuation can have a great impact in the rhythm, the overall structure and the versification of the song.

Once that is done, the data is tokenized using NLTK's word tokenizer and stored in a list of unique words. From said list of unique words, two dictionaries are made: one that maps each word to

an integer index, and a second one that maps each index to a word. This will be used when generating. After this, sentence windows and their target words are created. For every 10 words, the goal for the model will be to predict the eleventh, which is the target. 59,820 sentence-target pairs were created. Finally, numpy arrays that match characters to integers are made and then turned into Torch tensors, which will be loaded into train batches via a DataLoader.

The model of choice for this project is a simple LSTM with dropout to prevent overfitting and a fully connected layer to perform classification. The model is trained on 20 epochs and with a batch size of 32. The size of both the hidden dimension and the embedding dimension is of 256. The total loss of the training loop was of 0.07416 (rounded). Such a low loss would indicate that the model does not make many mistakes when predicting the next words. However, this is unfortunately not the case. Further discussion will be provided in the following sections.

To generate the songs, the first thing done is a function called lil_random. Its purpose is to prevent the most statistically likely word every time. Since it is unavoidable that the most common words, the words that appear more often such as connectors, determiners, some verbs, etc., will always be in the top prediction options, this function selects one of the few most likely relatively at random. In human language, not only the most likely to appear words appear in every sentence, after all. After this, a function called lyricmaker uses the model to calculate and output a song of 400 words of length (this parameter was chosen based on a general average of how many words Taylor Swift songs tend to contain). This function also splits the song into verses. Each song is produced from a starter line of 10 words, which are the first 10 words from one of Taylor Swift's actual songs.

Finally, to evaluate the natural language generated by the model, two methods have been used. The first one is human judgement and the second one is the Jury metric called Bleu. I decided on evaluating using this metric because I was not able to make BERTScore work on mltgpu, which was my first evaluation option. I got constructor and connection errors when trying to test my outputs, so Jury was my alternative. Bleu from Jury calculates scores based on the relation between the produced output and a reference sentence. When doing my evaluation, I compare the output songs with the actual song that the first line used to generate the song came from. Of course, the scores are expected to be low, as it is incredibly difficult for the model to output anything similar in content to the original song.

To judge the results from a human perspective, I sent some of the outputted songs to two of my friends who are Taylor Swift fans, as well as to some of my classmates, who are neutral about Taylor Swift. The results of the human judgement method will be commented in the following section, together with the results.

**Results**

Simply put, the songs generated by T(AI)lor Swift are not good. It is evident, just by taking a look at them (it is not even necessary to read the lyrics fully), that there is a lack of grammaticality and semantic logic. This is due to a combination of various factors, which will be commented below.

Firstly, there are some words and particles that are repeated excessively throughout all the songs, such as ‘’s’ (appeared 383 times in 5 generated songs), ‘think’ (appeared 128 times), ‘hate’ (appeared 94 times), ‘hope’ (appeared 126 times) and ‘talks’ (appeared 88 times), among others. This is probably due to a few words being notably more frequent than the others and shadowing the rest of the words which might also be common, just a little less likely to appear.

Secondly, the words tend to repeat themselves a lot, one after another. Some examples of this occurrence are “talks keep you talks talks talks talks talks hate hate”, “do do do” and “my hope think hope think and hope hope hope think”. I cannot be sure of the reason behind these repetitions, but I have two main hypotheses. The first one is that, even though one specific word has just been generated, the probability of the same word appearing is one of the highest simply because that word is commonly found throughout the whole dataset and it is still likely to appear, probability-wise. The second hypothesis is that Taylor Swift’s lyrics do have some word repetitions, and that is reflected in the performance of the model when predicting. One good example is the song ‘Shake It Off’ (2014), which contains lyrics such as “’Cause the players gonna play, play, play, play, play”, “And the haters gonna hate, hate, hate, hate, hate” and “Heartbreakers gonna break, break, break, break, break”.

Lastly, I realized once the model was put to use to generate lyrics that using words to predict the next word in the sentence would produce worse sentences than using characters to predict the next character in a sentence. My reasoning is that the probability distribution is spread throughout a lot less elements when generating character by character than when generating word by word. I had over 3.000 different words in my dataset whereas I would have probably had around 30 or 40 different characters. The overall likelihood of about 40 elements is more evenly distributed than that of over 3.000, and that might be why there are certain words that come up excessively, creating song lyrics that make no grammatical sense.

As said in the previous section, the results were evaluated using Jury’s metric Bleu (Bilingual Evaluation Understudy). This metric was originally designed to evaluate text that had been translated using machine translation. It computes the similarity between the original sentence and the output sentence. It is not exactly fitting for the evaluation of this task, but as the option to use BERTScore was discarded due to server problems, Bleu had to be used. The evaluation scores provided by Bleu in the five example generated songs of the project are very low: 0.031176 (rounded), 0.018633, 0.020490, 0.029375 and 0.020860. These results were expected, as the outputted songs are being compared to the original song where the seed 10 words that were used to initialize the text generation came from. It is

expected that the generated song will have some similarity to the original song. However, since it is just fed the first ten words of the song, getting a very similar song is very unlikely, hence the similarity scores provided by Bleu being low.

Finally, human judgement was used to evaluate the output product of my model. This human evaluation was done by simply showing some individuals one of the generated songs and letting them comment their thoughts freely. Some of these individuals are Taylor Swift fans, and others are not. Regardless of their likeness of Taylor Swift, all human evaluators agreed on the obvious fact that the results are not good, they do not make sense and they are funny to read out loud. When asked whether or not the lyrics reminded them of Taylor Swift, some said that the fact that some words are repeated did remind them of the singer, which might indicate that Taylor Swift might have some distinctive features. One of the distinctive features was, even if probably accidentally, caught by the model. Overall, all humans who judged the results conclude that the model is definitely in need of improvement if its goal is to be similar to Taylor Swift's work.

**Conclusion**

My main conclusion is that, may it be because of the limited dataset used to train the model, or the architectural decisions made in the different parts of the project, the text that is generated is not of good quality and there is therefore plenty of room for experimentation and improvement.

Dataset-wise, I believe that my dataset of choice did not contain enough data to train an NLG model. Although I had over 7.000 lines of song lyrics, these added up to 145 songs, which seems to not be enough for a machine to learn how to write songs.

The simple LSTM architecture that was chosen for the purpose of generating lyrics was probably not the best choice either. To improve the existing model, multiple extra linear layers could be added. Another improvement would be to add the option of bidirectionality in the LSTM layer, as words are not only dependant on the words that come before them, but also on those that accompany them right after.

It would also be interesting to try a whole new approach on generating lyrics, which is using a pretrained model of English, such as BERT or GPT-2, and fine-tuning it on Taylor Swift (or any other artist for that matter) lyrics. As suggested by Asad when I made my project proposal, probably the option that would yield better results would be to fine-tune an existing model of English in two steps: first, on lyrics in general (so, on a dataset of songs by various authors), and second, on Taylor Swift specifically. This way, we would be teaching a model that already knows the language to perform a specific language task. This is definitely an approach worth exploring.

As a final conclusion, I think that machines are still far from being able to supersede artists. Art is something very human and difficult to compute, therefore, machines are one (or more than one) step behind humans on this field. Although I believe that, by using the right data and the right model design the generated songs would probably be notably better than what my model is right now, art in general and music in particular are still something that humans do remarkably superior at making art.

I decided on this topic for the project because I was curious about a field of Language Technology that the students of the MLT programme had never worked on before, and I found this area of study immensely interesting and with a lot of different options to explore. I will definitely be trying to work with Natural Language Generation again in the future.

**Consulted works**

- https://www.tensorflow.org/text/tutorials/text_generation
- https://www.kaggle.com/super13579/let-s-auto-write-the-deep-purple-lysics-pytorch
- https://www.kaggle.com/anasofiauzsoy/writing-hamilton-lyrics-with-tensorflow-r
- https://en.wikipedia.org/wiki/BLEU
- https://machinelearningmastery.com/calculate-bleu-score-for-text-python/