ID2223 - Project Report

Clickbait titles generator

Aurélien Blicq

I - Project description

In the recent years, some news websites have been called out for producing so called “clickbait” content. This content is generally low in quality, easy to produce in large volume and is wrongfully advertised by a title and a thumbnail that are sensationalists and misleading. This is mainly done in order to attract traffic and generate ad revenue.

For instance the website BuzzFeed is renowned to use such practices and is responsible for articles like “21 Images That Capture How Scary It Is Being In Sydney Today” or “Only An Actual Teacher Can Correctly Answer 6/8 Of These Grammar Questions”.

This trend has been so much exploited by some websites that it has become a meme on the internet, and is oftentime frowned upon.

For this project, we propose to implement a clickbait titles generator using LSTM. Since those titles are short texts and that there are a few templates that are usually used to mass-produce them (like quizzes, tops, …), we thought this data would be well suited for text generation.

II - Dataset

The Data used to trained the models comes in part from a dataset of 16 000 clickbait titles used by a research team to implement a SVM classifier that would discriminate clickbait titles and regular titles [1], and in part from the titles that I have been able to extract from BuzzFeed through their RSS feed (see rssFeedParser.py script). The latter method yielded a total of about 2500 unique title over the course of several weeks. In total, the dataset used is composed of about 18 500 titles (see clickbait_dataset.txt).

III - How to run

The code is provided in the project.ipynb file. Before running, make sure all the necessary libraries are installed (list in the first cell).

IV - Preprocessing

We first have to modify our titles dataset into an array of vectors, in order for our model to be able to process it. To this purpose, we use the Tokenizer class of TensorFlow.

First, we add the “<start>” and “<end>” words respectively at the start and the end of our words. When we want to generate a new title, we just input the “<start>” word to our network. When our network outputs “<end>”, we know it just finished to generate a title.

We only retain the 5 000 most used word in our dataset, in order for our model to have less parameters to train (see section V), and all the other words are marked with the special token “<unk>”. We also remove some special character (like brackets, quotes, commas, etc.) in order to isolate the words (otherwise, “car,” would be a different word than “car”). Finally, we use a special token “<pad>” to pad our titles and have them be the same length, so that our list of titles can be represented as a numpy array of dense vectors.

V - Model

In order to generate a title (i.e. a sequence of words), we need a model that takes a word as input and output the word that should follow.

For this project, a model containing three layers was used. First, a embedding layer is used to encode the input word (encoded as a one-hot vector in sparse representation), into a smaller embedding space. This allows for a more semantically accurate representation of the words, even more so here, where the weights of the embedding are learned at the same time as the network learns to generate new titles.

Then, a LSTM layer is used, which is where the magic operates. An LSTM is a recurrent neural network with a long term memory added to handle long term dependencies better, and they are known to be good at processing sequences.

Finally, a Dense layer is used to convert the output of the LSTM into a one-hot vector representing the word that should follow the input.

In this model, the more words we have in the dictionary, the more parameters we have in the Embedding and Dense layers. Which is why we restricted ourselves to 5 000 words in the dictionary.

VI - Training

In order to train this model, we used the categorical cross-entropy loss, and the Adam optimizer. We made a custom training function in order to use teacher forcing, that is to say that the network is corrected after predicted each word, as its next input is the true next word instead of the one it predicted.

The training was run on a laptop with an intel core i7 processor and a NVIDIA Geoforce GTX 950M graphics card running Archlinux.

The network was trained for about 30 epochs, each epoch taking approximately five minutes.

VII - Results

After this training, we are able to generate new titles by first inputting “<start>” token and feeding the output of the network back into it until it generates the “<end>” token. Here are some titles generated by the network:

```
oprah winfrey and ryan reynolds posed at on drag race uk
reminder that ariana grande can you name
17 people who totally owned new jays there will never want
the 1 and friends quiz you're basically school so controversial <unk> purrfect
adults keep two wall of under <unk> <unk> google and we'll guess your bodily preferences
this quiz will determine if you'll actually get rid of avril grande or <unk> <unk>
you'll only guess which k pop group from 2019 based on which bts you should visit what your crush
here are all the <unk> to taylor swift's way laugh the investigation into their lives
19 times bbc company was the best romantic on this decade
which character from facebook are you
62 christmas videos and we'll tell you where are 2020
```

As we can see, none of these titles really make sense. However, our network still picked some patterns.

We can see a lot of quizzes with formulations like “and we’ll guess your”, “this quiz will determine” or “which character from facebook are you”, that appear a lot in the dataset. In the same note, a lot of tops or lists are generated. So, our model did manage to pick up some of the patterns in the BuzzFeed titles.

Another thing to remark is that our network learned to produce the “<end>” token and the length of the generated titles is reasonable for a title.

We also see that our network learned celebrities’ names. It learned that after “oprah”, comes “winfrey”, or that after “taylor” comes “swift” or “swift’s”.

Finally, we can see that our titles are not so incoherent. One that mentions K-Pop in the beginning mentions BTS (a K-Pop group) at the end, or mentioning Christmas and the new year in the same title.

VIII - Conclusion

The model presented here yielded some interesting results. While not perfect, some hyper-parameter tuning, a broader dataset and a longer training made on a more powerful machine might be enough to improve significantly the quality of the generated titles.

IX - References

[1] https://github.com/bhargaviparanjape/clickbait