Clickbait classification and generation: A Kaggle competition hosted by MSCI641

Dip Patel
University of Waterloo
Waterloo, ON, CA
dv9patel@uwaterloo.ca

Abstract

In this project report, I present different methodologies and results on clickbait generation and classification task. For this project, candidate is supposed to carried out 2 sub-tasks. First is, given the posts gathered from different social media platform such as, Reedit, Twitter, task is to predict which clickbait generation should be suitable for given post. And, second task includes generation of clickbait using given posts. Dataset for this project is available on zenodo and kaggle. For, first sub-task, both machine learning and deep learning approaches are used. Deep learning, pre-trained transformer model out-perform other technique by far margin. In order to solve second sub-task, I used pre-trained sequence to sequence transformer. Training it from the scratch requires extensive compute resource and data. For this task, training data is limited. Therefore finetuning the pre-trained transformer model seems better approach to tackle the problem. For task-1, I achieved best accuracy score of **0.7753%, Which is greater than semeval2023 competition's best score of 0.74%. And, for Task-2's best Meteor score, I have achieved is 0.299**.

1 Background/Related Work

Clickbait is text, specifically designed in such a way that it attracts attention of mass users. A form of teaser, aim to expose "curiosity gap", by not providing enough information on link, therefore, if user wants to know more, then in that case user has to visit the link. The reason why clickbait is popular because it ultimately, drive more user to website. Consequently, increases CTRs(Click-through rates) of website. And modern search engines, more or less presents website based on its CRTs rates. Therefore, Clickbait increases the traffic, along with increased advertisement revenue. Clickbait is also popular choice on streaming platform, that particularly thrives on "targeted Ads". On such example includes YouTube, revealing at ICES(International Consumer Electronics Show) that most of the watched videos and watch-time is generated through personalised information and advertisement rather than google searches[1].

Before, the proliferation of Natural Language processing(NLP), clickbait is fabricated by human to catch attention of users. However, after the advent of deep learning in NLP, clickbaits can also be generated using deep learning models. In addition to that, Reinforcement leaning can be employed to further improves deep learning models on clickbait generation, by either providing Human Feedback or by providing reward based on the attention it(generated clickbait) grabbed on internet. Previous studies on clickbait generation using reinforcement learning include the work of Xu, Peng et al. [2]. Where they used something called "sensationalism score" as reward in Reinforcement learning. The way they calculated "sensationalism score" is by training a sensationalism scorer model, which classifying online headlines with clickbait against a headlines with baseline model generation from summarization model. This process somehow seems similar to GAN process, where adversary tries to classify whether generation by the model is accurate or fake.

In addition to this, some studies also includes, generation of clickbait for downstream classification task [3]. In their studies, their main downstream task is to detect whether particular headline is clickbait or not. In order to tackle this problem using supervised learning required to have large annotated dataset. Which is quite difficult because high cost involved in labeling. Therefore, they generated synthetic data with specific style using style transfer, which later used as data, feed into classification model for detection. Study concluded with improving detection of clickbait.

Generation of clickbait for post can be considered as "stylistic summarization". At one hand we want to summarized whole post in 2-3 sentences,

but with some style, so that it can lure reader. However, task of clickbait generation is quite difficult, because there are several challenges, such as, limited dataset available. Although, we can get tons of training data from the internet using social media, but filtering out clickbait and original headline requires some automated mechanism. But, to build filtering automated mechanism we requires cleaned annotated data in first place. So we are in kind of vicious cycle dependency, wherein to break that dependency, we require to have data at first place.

For this project, we have two sub-tasks to complete. First is, clickbait type classification (i.e., should the spoiler be a phrase, a passage, to have multipart). Second task is to generate clickbait for the given posts. Dataset for this task is available on Kaggle under clickbait detection MSCI641, S24. There is separate section devoted for the Dataset description. However, in brief, total 4000 dataset available. Which split further 3200,400,400 for training, validation and testing. So effectively, for actual training we have 3200 samples. Not a lot for generation task, but still we can use this data with transfer learning to get better result.

2 Dataset

Dataset is available on kaggle under clickbait detection MSCI641, S24. Which contains in total 4000 samples. Which further got divided into training, validation and testing by 80%,10%,10% respectively. Dataset is in JSON Lines (JSONL) format. Wherein, each line contains separate JSON object. There are several advantages associated with this format over CSV(comma-separated values) and XML(Extensible Markup Language). First is, JSONL can handles complex, nested structure. Which is not possible in case of CSV. Often, CSV is limited to flat, tabular data. We can handle nested structure but it increases overall complexity and often requires non-standard workaround.

Each JSONL object in the training and validation dataset has following fields.

uuid :- A unique identifier for each dataset entry.

postText :- The text of the clickbait post which is to be spoiled.

targetParagraph :- A paragraph manually extracted from the linked web page.

targetTitle :- The title of the linked web page.

targetURL :- The URL of the linked web page.

humanSpoiler :- Human generated spoiler of the clickbait post from its linked web page. This field is available during training and validation only.

spoiler :- Human extracted spoiler of the clickbait post from the linked web page. Also, this field is available during training and validation.

spoilerPosition :- The position of the human extracted spoiler for the clickbait post from the linked web page. Again, this is available only during training and validation.

tags :- Type of spoiler (*"phrase", "passage", "multi"*). This field is only available during training and validation, and tags should be predicted during task-1.

Table-1 contains brief overview of how dataset looks and what some important fields contains.

Apart from this, dataset entry contains some other fields as well, such as, postId, PostPlatform, targetDescription, targetKeywords, and targetMedia. After careful observation on different fields on dataset, I came to conclusion, that above mention fields are not consistent. Which means, for some dataset entry, those entries are available, while for other we don't have those entries.

for the task-1 and task-2, I utilised 'postText', 'targetParagraph', 'targetTitle' as input data. Which I find sufficient for generation and classification task. Because it contains all the necessary information to make accurate prediction. Working with this dataset is challanging, because size of the input is very large. For example, for some entries we have more than 5000 words. which is quite huge for today's deep learning models. There aren't any model till this date which can handle such a long sequence of data efficiently. For this reason either we have to reduce the size of the dataset by some mean or keep most useful data for model feeding while truncating rest of the data.

3 Task-1 : Spoiler Type Classification

Task-1 at first looks simple, but it is quite challenging. In task-1, we are given data text, data 'X', and task is to predict whether particular post should be spoiled as, Passage, Paragraph or Multiline. In dataset, author used the word tags. Therefore, for subsequent discussion, I will use the tags instead of labels, though labels is profound in literature.

UUID	Clickbait Post		Linked webpages		Spoiler	
	Platform	Text	Title	Paragraph	Type	Position
1	Reddit	Wes Welker Wanted Dinner...	Wes Welker Wanted Dinner...	It 2019 ll be just like old times...	passage	[[3,151],[3,186]]
2	Twitter	NASA sets date for full recovery..	Hole in ozone layer expected to make full..	2070 is shaping up to be a great year...	phrase	[[0,0],[0,4]]
3	Twitter	This is what makes employees happy..	Intellectual Stimulation Trumps Money For..	Despite common belief, money isn't ..	phrase	[[1,186],[1,210]]

Table 1: Glimpse of Dataset and its fields

Those tags were given by human, according to whom which type of spoil is best to attract more audience or which tag is more appropriate for particular post-text[4].

In order to solve this task, I used two approaches. First is to tackle this problem using machine learning algorithm such as, naive-bayes. And second approach is to use state-of-the art pre-trained transformer model for solving the classification problem.

3.1 Naive Bayes Model

Naive bayes model works on simple but sometime crude assumption that all words are indepedent of each other given the class type. Therefore, we can calculate probability of class given the text by following formula.

$$\begin{aligned} P(c|X) &= \frac{P(X|c) \cdot P(c)}{P(X)} \\ &= \frac{P(x_1|c) \cdot P(x_2|c) \cdot ... \cdot P(x_n|c) \cdot P(c)}{P(X)} \\ &\approx P(x_1|c) \cdot P(x_2|c) \cdot ... \cdot P(x_n|c) \cdot P(c) \end{aligned}$$

$$\boxed{\hat{y} = \underset{c_i}{\operatorname{argmax}}\, P(c_i|X)}$$

Now, why can we restrict our self to only finding the probability of unigrams. We can further extends this idea to bigram(s) and trigram(s). Considering bigram and trigram improves performance. Because our simple assumption in naive-bayes model is not always hold in real-life. Therefore, with this argument we can consider n-gram approach, where I consider sequence of n words rather than single word or pair of word. But considering n-gram has its own limit. First is, if we have size of vocabulary |V|, then considering all n-gram means in total we have $|V|^n$ rows, therefore, for higher values of n, problem will become intractable. Moreover, table will be sparse because only few meaningful sequences will have high probability while all others have zero or near zero probability.

Model	Hyperparameter	Validation Accuracy
GNB	unigram	0.4237
GNB	bigram	0.4262
GNB	unigram + bigram	0.425
GNB	tf-idf	**0.4512**

Table 2: Gaussian Naive bayes & its variation's results

For classification task, data has almost 90212 unique words. Therefore, I limit my study to unigram and bigram only. Table below shows that, what accuracy I got for both variations. By looking at the result in table-2, we can clearly say that, almost unigram and bigram model produce more or less same result. However with tf-idf vectorizer we achieved greater accuracy.

Accuracy score is low. But we can conclude that naive-bayes is learning from the data. Because, suppose we have model that randomly generating answer from [0,1,2] then expectation of accurate answer would be 33%. And here we are achieving results >33%, therefore we can say that model is learning from the data.

However, Accuracy score is low. Therefore our second trial is to use state-of-the art transformer model for classification task.

3.2 Transformer Architecture

Transformer Model, introduced by Vaswani et al. in their 2017 paper [5], "Attention is All You Need", represents significant breakthrough in the field of natural language processing. Transformer model employs noval architecture purely based on attention mechanism, unlike other traditional sequence to sequence architecture, which rely on recurrent neural network(RNN) or convolutional neural network(CNN).

Training transformer model from scratch is quite difficult. Because training of transformer model is slow and we don't have enough training data so that our transformer model generalized well on classification task. Consequently, for this two reasons, I decided to finetune pre-trained transformer model. Which comes with two great benefits. First is, majority of transformer model are pre-trained on large corpus of internet data. Therefore, They are generalized enough for classification task. Moreover, as they are pre-trained, for finetuning, it will effectively takes less time to converge. However, we need to take care of phenomenon called 'catastrophic forgetting'. Which essentially referred to a situation where finetuning pre-trained transformer losses its previously learned information.

Furthermore, There are two ways through which we can use pre-trained transformer. First, we can use it for finetuning. Which means we are essentially updating weights of transformer for downstream task (in our case is multi-class-classification). Second, we can use transformer output as embedding and use that embedding as input to classifier head, which will further predict type of class based on the input embedding. So essentially, during training, transformer weights are not getting updated, only classifier head's weighs are getting updated.

Apart from this, there is still one problem left to resolve. And, that is, how to deal with long text while we are working with transformer architecture. Majority of proposed transformer architecture has inbuilt limit of supporting 512 tokens at a time. In our both tasks, data includes 'postText', which is of course more than 30 sentences (approx. 550 tokens). In order to dealt with large input text, I employed various strategies and compare its results to come at conclusion about which works better for given tasks.

3.2.1 Various strategies used in this study for dealing with large input sequence

Truncation :- In this strategy, we just truncate the data once we hit the limit of passing information to transformer architecture. The reason why we use this approach is because this approach works as baseline for comparison.

ROUGE Score :- According to the Wikipedia definition of ROUGE (Recall-Oriented Understudy for Gisting Evaluation), is set of metrics used to evaluate automatic summarization in natural language processing. ROUGE metrics ranges between 0 to 1, Where 0 indicates lower similarity and 1 indicates higher similarity.

In my study I used rouge score to find most informative information from the input sequence. And use only that information in training objective. The strategy, which we used to extract information is similar to the one used in "Pegasus: pre-training with extracted gap-sentences for abstractive summarization." [6] pre-training objective(Zhang, Jingqing et at., 2020).

Text Rank :- Text Rank algorithm is very similar to page rank algorithm [7] used for page ranking. In text Rank algorithm instead of web-page we are using sentences from the given text. So pipeline goes something like this. first we tokenize sentences from the given text. Once, sentences are tokenized, we need to vetorize whole sentence. For that we can use different approach. For this particular study we used tf-idf vectorizer for converting sentences to vector. Second, we need to find similarity between to sentences. For that cosine similarity mostly used. Once we find similarity then we can apply page rank algorithm to get rank for each sentences. And once we had rank for each sentences, we can easily extract top-k rank sentences from the data as input to our model.

BERTSUMEXT :- So far, we have traversed different unsupervised learning approach in order to extract more informative data from the postText and then feed that data to our training model. Different unsupervised learning approach includes, applying rouge metric score, Text-rank algorithm to extract meaningful information. But unfortunately this approach did not give good result. After carefully analysing, these unsupervised learning

method, I came to an conclusion that there are two problem associated with these unsupervised learning approach to extract meaningful information.

1st problem is, as those methods are rely solely on finding similarity across sentences. And when they find similarity it gave good score to those sentences. And similarity measured based on word-level similarity rather than context level. therefore, essentially, what we are doing is feeding same type of information to downstream task. Which is problematic because, now model has less diverse set of information then before and I think that is the reason, why extracting meaningful information using unsupervised learning method did not work well in comparison with naive truncation.

Still there is one approach left to test and that is extractive summarization technique using BERT model. This particular approach is known as BERTSUMEXT [8]. Where pre-trained BERT, trained on language modeling and next sentence prediction task is attach with classification head that classify which sentence is relevant and which is not. In that way, we can extract all relevant sentences from the paragraph and form an extractive summary of that paragraph. In order to train this model, we used CNN/daily dataset[9]. For this task I have not specifically trained model, rather I used already trained model for this task from huggingface [eReverter/bert-finetuned-cnn_dailymail].

BERTSUMEXT will give score for each sentences about its importance in paragraph. Suppose, we have three sentences in our paragraph [sent1, sent2, sent3], then BERTSUMEXT will give score for each of this three sentences. Let's suppose scores are [0.9, 0.3, 0.6]. Highest score means more important. Therefore, looking at scores, sent1 looks more important than other. So in this way we can gather more important information from the paragraph. In order to gather important information, I used three different approaches.

1. Naive approach of selecting top-k sentence, which has the highest score among other sentences.

2. Selecting continuous sequence of k sentences, which has highest cumulative score among other continuous sequences.

3. Selecting sub-sequence(non-continuous) of k sentences, which has the highest cumulative score among other sub-sequences.

Each process has their own merits and demerits. Let's discussed them one-by-one. By selecting top-k sentence, of course it increases the information feed to the downstream task but certainly at the cost of losing sentence level continuity. Previously, we used different approaches wherein we fed top-k sentences, did not yield good result score. The only reason we are applying it here is to compare its result with rest of the two approaches. Second approach includes, selecting continuous sequence of k sentences which has highest cumulative score. This approach seems like balancing the benefits of both worlds. First, it preserves sentence level continuity, whereas on the other hand it also extract informative information. However, there are one drawback associated with this approach, where sometime it might possible that most informative isolated information get left out. This is the cost we are paying in order to preserve sentence level continuity. Last approach is one of the famous dynamic problem in algorithm. Where our task is to find longest sub sequence, having highest cumulative score. This approach seems in middle of above two approach. Neither we are missing isolated information nor we are hurting continuity much, as we are in selecting top-k sentences.

3.3 Result

For this study we used two different transformer architecture, BERT [10] and RoBERTa [11]. Initially, I play with this two model with different hyper-parameter setting. Result of this is present in table-3. By looking at the result, I came to conclusion that RoBERTa is outperforming BERT model. And, finetuning works better in comparison with embedding. Consequently, for further experiment with Page rank and BERTSUMEXT, I specifically adhere to RoBERTa model with finetuning. Which results are presented in table-4.

BERT trained on book corpus and English Wikipedia where as RoBERTa is trained on much larger dataset. Therefore, in term of generalization RoBERTa is better compare to BERT. Apart from this BERT is pre-trained on Masked Language Modeling(MLM) and Next Sentence Prediction(NSP) tasks, whereas RoBERTa only focused on MLM task. In addition to this, in MLM task BERT used same masked token throughout the training. on the other hand, RoBERTa implements dynamic masking. Lastly, BERT used wordPiece embedding in contrast with RoBERTa, where au-

Model	Hyperparameters		Validation Score
	Inputs	Finetuning/Embedding	
BERT	Normal Truncation	Embedding	0.50
BERT	top-k Rouge sentences	Embedding	0.49
BERT	Normal Truncation	Finetuning	0.55
BERT	top-k Rouge sentences	Finetuning	0.53
RoBERTa	Normal Truncation	Embedding	0.41
RoBERTa	top-k Rouge sentences	Embedding	0.47
RoBERTa	Normal Truncation	Finetuning	**0.74**
RoBERTa	top-k Rouge sentences	Finetuning	**0.65**

Table 3: Result for Task-1/using BERT and RoBERTa with "top-k rouge scored sentence as input" strategy

Model	Hyperparameter	Validation Score	Test Score
RoBERTa	top-k Textranked sentences	0.6937	0.6867
RoBERTa	top-k BERTSUMEXT scored sentences	0.5675	0.5878
RoBERTa	**sequence with maximum cumulative BERT-SUMEXT score**	**0.74**	**0.7753**
RoBERTa	subsequence with maximum cumulative BERT-SUMEXT score	0.74	0.7599

Table 4: Result for Task-1 using RoBERTa with TextRank and BERTSUMEXT

thor decided to used Byte-Pair Encoding(BPE) similar to GPT-2.

Studies showed that RoBERTa performed better than BERT model across different NLP task. Therefore, it is preferred to RoBERTa over BERT. However we used both the model and cross-validate both the results.

All model training for different approaches is **limited to 5 epochs** with **standard learning rate of 1e-5 - 2e-5**. Furthermore, For training we used Pytorch 'deep learning framework' of python. And, training is carried out on Nvidia-T4 GPU.

4 Task-2 : Spoiler generation

For task-2, we are asked to generate spoilers for the given post. In the data, we get different fields which we can use it as input data to generate spoilers, but I used only those fields which are common across all inputs and have relevance in spoiler generation. Same strategy as what I used in task-1. For spoiler generation task as well, we are dealing with the same problem of input size.

In Natural language processing, theoretically family of RNN(Recurrent Neural Network) has capability of modeling any length of sequence. And according to universal approximation theorem, RNN with single hidden layer can approximate any function. However, practically family of RNN often face problem of vanishing and exploding gradient. Though we can employ different strategies to counter this situation, not a single method is effective and guaranteed to give better results.

On the other hand, Transformer models are employed to address various challenging NLP tasks. And it consistently outperforming existing models. However, transformer is powerful and showed great performance across different NLP task, it cannot handle large sequence after certain length. Majority of state-of-the-art models available can handle size up to 512 tokens. In addition to this, majority of the transformer models such as Bert, Bart and GPT are trained on entire internet. Entire internet corpus has enormous amount of vocabulary, in the order of billions. Handling such a large vocabulary is difficult. Therefore, tokenizer such as BPE(Byte Pair Encoding), wordpiece[12] is used, Which basically break down words into subword units. On an average, the ratio around 1.2 to 1.5 tokens per word. Consequently, At a time transformer model can handle 341 to 426 words, almost 30 to 40 sentences. Which is enough for most of the NLP task but for summarization task it is often not enough. We can consider clickbait generation task as stylistic summarization, where

Model	Hyperparameter		Meteor Score for Testdata
	Inputs	Generation Type	
BART	top-k rouge scored sentences	Normal	0.1517
BART	Normal Truncation	Normal	0.1605
BART	top-k rouge scored sentences	Conditional	0.1743
BART	Normal Truncation	Conditional	0.1928

Table 5: Result for Task-2 using BART as seq to seq generator

Model	Hyperparameter		Meteor Score for Testdata
	Inputs	Generation Type	
BART	top-k BERTSUMEXT scored sentences	Conditional	0.1928
BART	Sequence of Maximum BERTSUMEXT scored sentences	Conditional	**0.2999**
BART	Sub-sequence of Maximum BERTSUMEXT scored sentences	Conditional	**0.2950**

Table 6: Result for Task-2 using BART as seq to seq generator with BERTSUMEXT as sentence extractor

we want to generate short catchy summary of given post.

In order to calculate relevance of each input sentence, we used the same approach that we used in task-1. However, for generation task, we used BART[13] model as sequence to sequence generator.

4.1 Results

For this task, I used BART as generator and conditional generator. In conditional generation, along with input, I provide what type of spoiler is expected from the generator to produce. By comparing result of generation and conditional generation, condition generation works better. Table-5, contains results, I achieved for the task-2. Looking at the result from Table-5, we can clearly say that, normal truncation works better and yields better result in comparison with rouge score for sentence extraction.

Result of study showed that conditional generation works better in comparison with normal generation. Therefore, for further training, I only consider conditional generation aspect. Table-6, contains result that I got, using BART as conditional sequence to sequence generator, where BERTSUMEXT used as sentence extractor. Furthermore, I used three different strategy to extract sentences. Each having its pros and cons, Which we have discussed above in section 3. Turns out that, BERTSUMEXT strategy overall works better. Apart from this, when we extract sentences which has maximum consecutive score yields better result, evident from the results.

5 Code

All experiment were conducted using the scripts/notebook provided in my repository at Clickbait-1 and Clickbait-2.

6 Conclusion

This competition includes two task, Text-Classification and Text-generation. As a size of dataset is limited, to effectively solve this problem we choose finetuning a LLMs. As those LLMs are pre-trained on large language corpus, it provides benefit of better generalization. For Task-1 we used different strategies to solve classification task, and we got accuracy of **0.7753**. Which is higher than semeval-2023's first submission. For task-2, unfortunately we did not achieve high meteor score for generation task. But, almost **0.3** is consider to be good score.

For the future improvement perspective, to improve performance on task-2, we can use question answering strategy to generate(find-out) spoiler from the given post-text. Moreover, Another approach which might improve result is, unsupervised pre-training of transformer model on given dataset. Which can potentially improve results as now model has more contextual knowledge about the task at hand. Unfortunately, I could not touch

upon that aspect of learning. But unsupervised pre-training on given dataset looks promising.

References

[1] "YouTube Says 70% of All Watch Time is Driven by Its Own Recommendations".www.tubefilter.com. 11 January 2018. Archived from the original on 24 May 2024. Retrieved 29 April 2021.

[2] Xu, Peng, et al. "Clickbait? sensational headline generation with auto-tuned reinforcement learning." arXiv preprint arXiv:1909.03582 (2019).

[3] K. Shu, S. Wang, T. Le, D. Lee and H. Liu, "Deep Headline Generation for Clickbait Detection," 2018 IEEE International Conference on Data Mining (ICDM), Singapore, 2018, pp. 467-476, doi: 10.1109/ICDM.2018.00062. keywords: Generators;Media;Training;Recurrent neural networks;Decoding;Social network services;Training data;Data augmentation;Deep generative model;Clickbait detection

[4] Hagen, Matthias, et al. "Clickbait spoiling via question answering and passage retrieval." arXiv preprint arXiv:2203.10282 (2022).

[5] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[6] Zhang, Jingqing, et al. "Pegasus: Pre-training with extracted gap-sentences for abstractive summarization." International conference on machine learning. PMLR, 2020.

[7] Sergey Brin and Larry page. Google search engine. "http://google.stanford.edu."

[8] Liu, Yang. "Fine-tune BERT for extractive summarization." arXiv preprint arXiv:1903.10318 (2019).

[9] Nallapati, Ramesh, et al. "Abstractive text summarization using sequence-to-sequence rnns and beyond." arXiv preprint arXiv:1602.06023 (2016).

[10] Devlin, Jacob et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” North American Chapter of the Association for Computational Linguistics (2019).

[11] Liu, Yinhan, et al. "Roberta: A robustly optimized bert pretraining approach." arXiv preprint arXiv:1907.11692 (2019).

[12] Google. 2018. The WordPiece Algorithm in Open Source BERT. URL: https://github.com/ google-research/bert/blob/master/tokenization.py#L335-L358. Retrieved on 12/01/2020.

[13] Lewis, Mike, et al. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension." arXiv preprint arXiv:1910.13461 (2019).