Assessing Humor in Edited News Headlines with LSTM

Siyuan He

Abstract

This project aims at solving an regression problem of assessing the level of humor in edited news headlines with some atomic changes to original headlines. The problem is building a model to predict the degree of humor, given the original headline and the edited headline. The method we attempted to use is mainly bi-directional LSTM model with a sequence of specialized data pre-processing. We evaluate our result with rooted mean square root, and our result successfully beat two simple headlines including linear regression and mean of training scores.

1 Introduction

Many research have been done on how to judge whether some words is funny in a text, however, the humor doesn't always come from the meaning of word itself. There could be many cases where changing a non-funny word to a non-funny word can make the whole sentence funny. The phenomenon is interesting because the humor lies in the edition on a context instead of in the humorous meaning of the word itself. What we might further curious on is how to, or how short, or how simple the change could be to change a level of funniness of a context. The questions all lead us to discover the effect on the edit action to a context.

News headlines are always places where the technique of atomic changes leading to humor is used. The editor wants the headline to attract as more as attention as possible using humor, but the headline can't be modified much or the idea may be misunderstood. Hence, news headline provides with us a good sample for the research.

In this project, we're trying to build a model to predict how an atomic edit action, i.e. changing one word, can influence the level of humor of whole sentence. In more details, we quantify the level of humor of a sentence to a range from 0 to 3. The model takes a humorless sentence and a word to replace, and output the number representing the level of humor after edition.

The topic is a relatively new one within natural language processing tasks. The first related paper is published in 2019, and there is an ongoing online competition on the topic. Indeed, this project is aiming at attending the online competition based on the idea raised by (Nabil Hossain and Gamon., 2020). Since this is a new topic to research, we can only find very few related work, meaning that we're cutting edges.

We first attempted to use various model with same pre-processing, but the performance is even similar to simple baselines. One partial conclusion of our attempt is that data pre-processing is very important in this topic. If we can't use the information of difference in original sentence and edited sentence to encode the edit action, whatever model we choose may not result in a good performance.

Then we focus on the data pre-processing. We tried several pre-processing method in the problem besides common pre-processing, including double normalization, lemmatization, and low frequency word dropping. However, they tend to have only limited effect on the performance. With the final sequence of data pre-processing, we successfully beat the baseline, though only a little bit ahead.

If the topic can be further researched, the newspaper editors may feel happy in certain research on how edit actions can change humor, because the research can give them a tool to analysis the effect their edition may have on the humor of headline. More specifically, the research provides them with a tool to predict the level of humor by inputting a original headline and a edited headline.

2 Problem Definition and Data

2.1 Problem

At the top level, the problem is to predict the grade of funniness of the edited headline given the original and edited headline. To solve the problem, I'll build a model which takes a original news headline and a substitution on the headline, and the model gives a real number from 0 to 3 indicating the grade of the funniness.

More technically, the problem for this project is a regression program, because we want to output a value ranging from 0 to 3. The material is a set of original news headlines with an atomic change for each news headline provided individually, where an atomic change means substituting a word (or a short phrase) in the original headline. Each sample in the material is given a level of humor for the substitution ranging from 0 to 3. For example, one sample in the dataset could be "Kushner to visit *Mexico* following latest Trump tirades, therapist, 2.8", while the first element is the original headline, the second is the substitution and the last is the grade (Hossain et al., 2019). The substitution means to replace "therapist" to "Mexico" in this example. The task is to do regression on the given dataset and predict the score of funniness of a substitution of some headline.

Unlike normal natural language processing problems, we don't really care about the original sentence and the edited sentence, instead we care about how the edition can make some difference. For this reason, an intuitive idea is to "encode" the edit action to some trainable value, which is the actual difficulty of this problem.

2.2 Data

Since this project is from an online competition, so the dataset is obtained from the website and the related paper. More particularly, the dataset is introduced in paper (Hossain et al., 2019) and obtained in the website of competition `https://competitions.codalab.org/competitions/20970#learn_the_details-overview`. This is a dataset of the funniness grading of a edition from the original news headline.

The training dataset contains a 9652 lines of training data, which contains several labels. The first is id, indicating the id of the original headline and is not in some certain order. The next column "original" is the original news headlines. The third column in the substitution word, and the word to substitute in the original headline is labeled in "</>". The "grade" column seems useless and didn't mentioned in the document and the last column "meanGrade" is the grade score. Here's a sample cut of the dataset in table 1, and I think the dataset clearly express how the problem is defined.

The dataset is quite clean, which is saying that each line in the dataset is a good example to train. Since the data is newspaper headline, so we can expect no spelling error or causal tone inside. The score of most training set is distributed at 0 and 1, which means a lot of edit is not so funny. One potential problem for that could be we can't extract enough feature to determine "what is greatly funny" due to lack of high score examples.

3 Related Work

Since the project is directly from a competition, hence the paper related to the competition are the most important papers to the project.

The paper "“President Vows to Cut <Taxes> Hair”: Dataset and Analysis of Creative Text Editing for Humorous Headlines" describes the idea that we can analyze the edit action instead of word itself (Hossain et al., 2019). It also provides the dataset and evaluation method to the project, and the author of the paper also launches the competition project on the website (Nabil Hossain and Gamon., 2020). The paper gives me motivation to use LSTM model.

Word2vec is a necessary technique applied in our solution. However, training such as a model by ourselves is not efficient to because it takes a lot of time and the performance is not guaranteed. We choose to use google pre-trained model instead (Google, 2013). The model is trained on the GoogleNews, which is even the similar source to our problem.

Here's another attempt to find the level of humor from the analysis on words done by Tomas Engelthaler and Thomas T. Hills (Engelthaler and Hills, 2018). The author conducted a survey to let human label the level of humor in words. This work gives me some basic understanding on the humor, especially at the word level. Besides, it also provides me a ready-to-use data if I want to make use of the fun level on words in the model, however my attempts show that it's useless in the problem so I skip the idea.

The paper "What makes some word funny?"

id	**original**	**edit**	grades	**meanGrade**
14530	France is ‘ hunting down its citizens who joined <Isis/> ’ without trial in Iraq	twins	10000	0.2
13034	"Pentagon claims 2,000 % increase in Russian trolls after <Syria/> strikes . What does that mean ?"	bowling	33110	1.6
8731	Iceland PM Calls Snap Vote as Pedophile Furor Crashes <Coalition/>	party	22100	1.0
76	"In an apparent first & Iran and Israel <engage/> each other militarily"	slap	20000	0.4

Table 1: Samples from Dataset

(Westbury and Hollis, 2019) is a related work to the previous one. In the work, the author analyzed various aspect of deciding a word to be funny, and in the paper they're able to predict the human norm of the funny level of a word. However, the work still starts from the word level, not based on how a word with the context.

A Valitutti, A Doucet and H Toivonen's paper is related to the substitution generates human humor(Valitutti et al., 2013). In the work they generate human humor by substituting words in the context using some lexical constrains. However, not only lexical constrains generate humor, and our problem is not a generative problem. Hence the paper doesn't provide solid solution or direction to the problem, but it gives me quite a few motivation on how to design the model.

4 Methodology

All the codes can be found on my github `https://github.com/sweetsinpackets/SI630-Project`, and I think it might be better to directly read the code. I'm sorry for the messy code and file structure sincerely. There are many annotations in the file, which is enough for you to understand what's going on (though some are unnecessary and even out-dated).

To solve the problem, I design a sequence of pre-processing, word2vec, sentence vectoring, and bi-directional LSTM model. Besides usual NLP libraries including *sklearn, kersa, nltk*, I apply a Google pre-trained word to vector model (Google, 2013). We can easily load the model and use it like a dictionary, aka if we give a index as a string, it will return a 300-dimension vector. Since the model is quite large (indeed 3GB), so I didn't store it inside the github.

The main aim for pre-processing is to transform the given data into trainable vectors. Since the source is from newspaper, so it's unnecessary to apply spelling check on it. We first add a column to the original dataframe, where the content is replaced sentence. We use regular expression to detect the word to be edited (labeled with "</>") and replace by edition word. Then we remove every symbol and number in the sentence because they don't provide humorous meaning. Here we further turn all word into lower-cases and remove stop-words, which are quite usual in NLP tasks. We assume that rarely appeared words won't contribute much to humor because people are not very familiar to them. With that guess, we decide to drop a percentage of words appeared with lower frequency among all training set. We tested the performance among percentage $[0.1, 0.05, 0.2, 0.3]$, and we choose 0.1. It means we first count and rank the frequency of all words, and drop the last 0.1 percent form all sentences. After that we recover all transformed words into its original form, for example "trained" (past tense) to "train". Finally we tokenize sentences into a list of words and stored in a new column of the dataframe.

In the word2vec process, the aim is to find a vector to represent words. It's hard for us to train such as model ourselves, because we have no proper data and it takes quite a long time. As mentioned, we directly use google pre-trained model (Google, 2013). You can find more instructions in the website. Then we attempt to use a single vector to represent the whole sentence. After several attempts, we finally choose to simply use the mean of all words, and the reason will be discussed later.

We apply a bi-directional LSTM model to solve the problem. The model structure is an embedding layer with output dimension 1, one bi-directional LSTM layer with output dimension 1 and one 1-dim vector to scalar dense layer. The embedding layer serves as the first layer of the whole model, and it doesn't really change the dimension of the input. We add this layer because it may sometimes assert an error if we drop the layer, and adding such a layer can slightly enhance performance. The most important layer is bi-directional LSTM layer, and we use *tanh* function as activation. The dropping rate is set follow suggestions from website on the topic of applying LSTM on regression problems.

Layer	Output Size
Embedding	(None, None, Vector Dim)
Bi-LSTM	(None, 600)
Dense	(None, 1)

Table 2: Actual Model Structure on Training

Here's a trick on the dense layer. Since our problem specify the range to be $[0, 3]$, we use sigmoid and scaling to achieve that. When a number passes through the dense layer, it will be mapped into $[0, 1]$. Thus we scale all scores by dividing 3 before training, and multiply 3 after the prediction. For this reason, the MSE in the log files are not real MSE, but the mean square error on scaled number.

When we're training, there's a problem that the prediction are constant. To solve the problem, we further normalize the vector before they're thrown into the model. The normalize aims at making the variance larger by scaling into a large scale.

Finally, the result is evaluate through root mean square error (RMSE). Note that a lower RMSE means higher performance.

5 Evaluation and Results

5.1 Baseline

I set two baselines for the project. The first one is making all prediction as the mean score of training set. This baseline is also provided as the baseline of online competition, and even ranked middle at the online competition. Actually it even serves better than my own baseline requiring more knowledge, which is a linear regression model without any pre-processing (except for necessary operations such as replacement). It means I directly replace edited word into the sentence, put everything into word2vec, and put every word for linear regression. Though the training takes a lot of time (thanks to the small size of training set, it's acceptable), the result is much worse than the simpler baseline.

Baseline	RMSE
mean of score	0.578
direct linear regression	0.710

Table 3: Baseline

Though it seems strange that a more complex baseline is even worse than a baseline don't need knowledge on test data, I think it's reasonable to take the simpler baseline as the baseline of my project. The reason why the simpler baseline is better will be discussed later.

5.2 Result

The best score among my all attempts have a $RMSE = 0.559287$, smaller than the baseline, so I can claim that this project is succeeded (at least in performance). Here's a figure showing the performance at first 5 iterations of different models, and note that the MSE is not actual MSE, but the MSE on scaled data instead as mentioned.

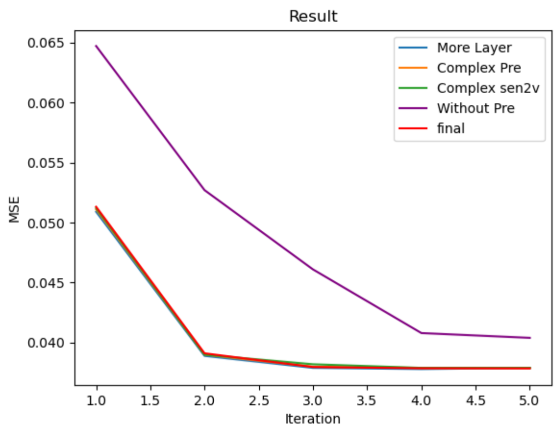

Figure 1: Result of Different Attempts

The performance of a model with only simple pre-processing is obviously worse than those models with more complex data pre-processing. However, it seems using even more complex data pre-processing (i.e. dropping some 0-score lines to form a balance training set) doesn't further enhance the performance, so we drop some unnecessary complex data pre-processing.

The performance of the model with more complex method to calculate sentence representation seems not increase the performance. Strangely, the performance is even slightly worse than the simple mean method.

Adding more layers seems no help to the performance as well. More layers of bi-directional LSTM have lower training loss, but it's worse on validation loss. We regard it might be a problem of over-fitting.

6 Discussion

Though my model beat the baseline, but I don't think it performs well. Indeed, if I'm a target user, I would never want to try such a model with the result. We observe that most of our prediction is still in the range $[0, 1]$, which means the model would fail for almost all actual humor cases. From the result, we can claim that our model didn't actually learn how to judge whether a edition is humorous or not. It also indicates my model didn't really beat the baseline (though the performance is better), because they both achieve better performance by achieving high score on non-funny cases.

I think the most important reason is that we don't really make use of the information of "edit action". The way we build our model is like predicting the funniness of the edited sentence, with almost no use on the original sentence. However, I've tried to use the information of location in the sentence to vector process by assigning higher weight for words near the edition, but the performance is even worse. Hence, I think we need to think about a new way to encode the action to some trainable vector, or we can't really solve the problem.

In addition, it's strange that a simple baseline (mean of score) would be better than a linear regression baseline. We think the problem is because both training set and test test are not balanced. The score 0 and 1 occupies most of the samples, hence the dataset is obviously biased on not or less funny samples. Hence the mean of all training scores happens to locate within the proper range of the distribution. However, I would regard the linear regression as a model really learns something, because it successfully predict some cases of actual humorous samples, and we can observe obvious variety among all its prediction.

7 Conclusion

In this project, we attempt to build a model to solve prediction the funniness of an atomic edit action on a not funny sentence. Though our performance beats the baseline we set, but the model is still far from satisfactory. From our attempts, we conclude that solving such a action-related problem requires a proper way to encode the action to some trainable features. However, in this project we failed to develop such an encode, so our model fails to predict most cases with real funny edit actions. Thus we think the focus for future work should focus on finding a proper structure to encode the action.

All the code has been published on the github `https://github.com/sweetsinpackets/SI630-Project`.

8 Other Things We Tried

We have mentioned that we add the embedding layer because dropping it may cause some error during training. We tried to find and solve the problem, but we failed to do so. The reason is that the error doesn't appear every time (approximately once for 4 training) ,and the time for each training is almost an hour. Hence, we think it's too time consuming to get rid of the problem, so we keep the embedding layer.

9 What You Would Have Done Differently or Next

If I need to start the project again, the most thing I would do differently is I won't spend a lot time on network choosing. Actually, I tried a lot of different configures on the network structure, but they work similarly worse. Finally I realized that the problem lies in the fact I didn't properly use the information, but it's too late to re-design the structure. Hence, the suggestion I derived from my work is to think carefully on the limit of current popular models if you encounter a new type of problem.

10 Work Plan

The work plan proposed in proposal (and in reality) is:

- Before update: Write functions to get and store training data, apply necessary preprocessings.
- Around Update: Design the structure of network (take most time).
- Implement and store the final model.
- Predict and evaluate.
- Repeat on other models.

However, it seems I misunderstand the actual problem of the tasks. The pre-processing and design is the core, while the model is relatively easier. Hence I would like to adjust the work plan to:

- Before Update: Write functions to get dataset and necessary data cleaning. Write functions to do word2vec and train baseline.
- 3 weeks: Design the data structure and deeper pre-processing.
- 4 days: Train on a proper model
- Write final report.

Acknowledgments

I acknowledge the SI630 teaching group for suggestion on this project topic selection and opinions on my previous ideas.

References

Tomas Engelthaler and Thomas T Hills. 2018. Humor norms for 4,997 english words. *Behavior research methods* 50(3):1116–1124.

Google. 2013. "googlenews word2vec pre-trained model". `https://code.google.com/archive/p/word2vec/`.

Nabil Hossain, John Krumm, and Michael Gamon. 2019. “president vows to cut <taxes> hair”: Dataset and analysis of creative text editing for humorous headlines. 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)*. Association for Computational Linguistics, Minneapolis, Minnesota, pages 133–142. `https://doi.org/10.18653/v1/N19-1012`.

John Krumm Nabil Hossain and Michael Gamon. 2020. Semeval-2020 task 7: Assessing humor in edited news headlines. In Proceedings of International Workshop on Semantic Evaluation (SemEval-2020).

Alessandro Valitutti, Antoine Doucet, Hannu Toivonen, Jukka M Toivanen, et al. 2013. " let everything turn well in your wife": Generation of adult humor using lexical constraints. In *The 51st Annual Meeting of the Association for Computational Linguistics (ACL) Volume 2: Short Papers*. Association for Computational Linguistics.

Chris Westbury and Geoff Hollis. 2019. Wriggly, squiffy, lummox, and boobs: What makes some words funny? *Journal of Experimental Psychology: General* 148(1):97.

A Supplemental Material

Please refer to `https://github.com/sweetsinpackets/SI630-Project` for the full code and probable further update. If you have some suggestions, please also leave an issue.