COSC 572 - Empirical Methods in NLP

Zirong Chen, Haotian Xue, Yuansheng Xie

May 9, 2020

Github Repository: https://github.com/RexZChen/EMNLP

Predicting the Funniness of Edited News Headlines

Zirong Chen
Dept. Computer Science
Georgetown University
Washington, D.C.
zc157@georgetown.edu

Haotian Xue
Dept. Computer Science
Georgetown University
Washington, D.C.
hx82@georgetown.edu

Yuansheng Xie
Dept. Computer Science
Georgetown University
Washington, D.C.
yx131@georgetown.edu

I. ABSTRACT

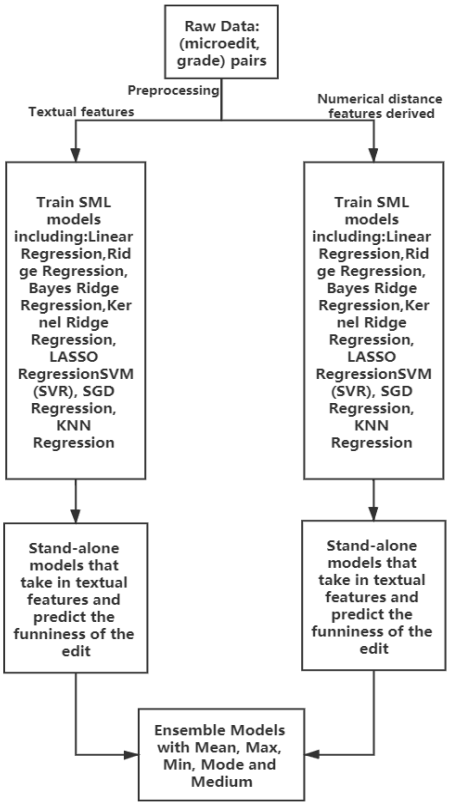
Using the *Humicroedit* dataset, a dataset for research in computational humor released by the University of Rochester [1], we build systems to detect and quantify humor in edited news headlines. We extract numerical and textual features to build Statistical Machine Learning and Deep Learning models that predict a grade of the funniness for each headline in our dataset. We evaluate our models using Root Mean Squared Error and gauge our potential placement on the leaderboard of SemEval Task 7, which, at this time, is within top 20.

II. INTRODUCTION

Humor detection continues to be a challenging AI problem. Through examining literature, we found combining computational linguistics with theories of humor to be an interesting area for experimentation. In this project we give our best shot at building an automated humor assessment system that detects and quantify humor using a carefully curated dataset and machine learning models. The dataset, *Humicroedit*, consists of samples of edited news headlines along with grades corresponding to the funniness of each headline. This dataset is thus especially helpful in our endeavor of utilizing supervised learning approaches to tackle humor detection and quantification. Using this dataset, we train stand-alone SML models as well as DL models to output a grade of funniness for each headline. We then ensemble the models we constructed by taking the mean, minimum and maximum of results of the models. We evaluate the performance of our models and the ensemble using Root Mean Squared Error (RMSE). We use this same metric to gauge our models and our potential placement on the leaderboard of SemEval Task 7. Currently, with our best model, we are placing within top 20. We would like to place a caveat here indicating that any speculations of our placement is, well, just speculations; because we do not have access to the actual validation data of SemEval Task 7. However, given that we are utilizing all the training data that was provided and testing on the provided testing set that our models were not exposed to during training, we have good reasons to believe that our speculations about our placements are within an acceptable margin of error to our actual placements, had we entered the contest.

III. ORGANIZATION

In this short section we present a flow chart that shows the flow and major components of our system. The specifics of the chart will become clearer as you read through the coming sections.



IV. DATASET

The *Humicroedit* dataset is created by the University of Rochester. The dataset contains edited news headlines and their corresponding grades of "funniness", where an edit is defined as the insertion of a single-word noun or verb to replace an existing entity or single-word noun or verb. It was noted that edits of other parts-or-speech were avoided because those edits did not provide enough variety of humor. [1] The news headlines were obtained using Google Big-Query from Reddit, specifically the sub-reddits of r/worldnews and r/politics, from January 2017 to May 2018. Mechanical Turk workers were then used for data annotation tasks which included making edits to generate humor and recognizing humor in edited headlines. The edited headlines' amount of humor, or funniness, are labeled on a scale of 0-3, with 0 being not funny at all and 3 being extremely funny. Each sample in the dataset consists of an original headline, a new word that is used to replace a word in the original headline, and the corresponding grade of the funniness of this edit. An example of a sample is given below; the word inside </> is the word being replaced:

Original headline	Replacement	Grade
Wisconsin Ironworker Challenges Paul Ryan For <House/> Seat.	Bicycle	2.2

For our project specifically, we obtained 17900 such to for the purpose of training and an additional 3024 samples for testing. Our preliminary analysis on this dataset found that the mean grade of funniness among all samples was 0.9 with a standard deviation of 0.2.

V. PREPROCESSING

We organize our tasks under preprocessing into two categories: **extracting numerical features** and **preprocessing textual data**. We explain them in more details in this section. Our preprocessing techniques are expected to improve the general results for our regression task. [2][3]

V. i. EXTRACTING NUMERICAL FEATURES

We extracted several numerical features from the raw data to build SML models. Namely, they are:

1) The position of the replaced word in the original sentence
2) The cosine similarity of the original word and replacement word
3) The Euclidean distance of the original word the replacement word

We thought to extract 1) because, in our literature review, we found that longer jokes tend to be funnier or have are potential to be funny because they have extra information. [9] [10]. In addition, [7] also mentions that humor usually arrives near the end of a joke after a setup in the beginning part. We therefore decided that the position of the replaced word in the sentence is of interest to us in our analysis and should be included as a part of our SML models. We thought to extract 2) and 3) because other literature suggested that incongruity, or a violation of expectation, can also contribute to humor. [4][6][8] Hence we thought to quantify incongruity in our samples by measuring the difference between the original word and the replacement word in an embedding space. To this end, we utilized a 300D GloVe embedding and extracted the cosine similarity and Euclidean distance between the original word and replacement word in that embedding.

V. ii. PREPROCESSING TEXTUAL DATA

We modified the raw text of our samples before feeding them into our DL architectures. Below are the several steps we took in this process as well as several steps we chose to forgo and our reasons for doing so:

+ We normalized all text to lower case
+ We removed all punctuations.
+ We removed all non-alphanumeric characters.
- We did not stem the words in our headlines, because, in our initial experimentations with our baseline models, we found that stemming did not improve the performance of our models.
- We did not lemmatize the words in our headlines because, similarly to stemming, we had found that lemmatization did not help improve the performance of our models.

VI. STATISTICAL MACHINE LEARNING

We trained several SML models that took the extracted numerical features as input and produced a funniness grade as output. [11] We go through each one in this section in more details.

VI. i. LINEAR REGRESSION

A Linear Regression model is always the first choice to implement in one regression task. So, we first implemented LR model as our baseline model and it achieved 0.587 RMSE. We then created a Ridge regression model and LASSO regression model, essentially applying L2 and L1 Regularization to the baseline model, but these approaches did not improve the results significantly. Ridge Regression (alpha=0.5) model achieved 0.587 RMSE and Lasso Regression (alpha = 0.6) model achieved 0.594 RMSE. Inclusion, we had found that adding regularization to simple linear regression did not help to lower RMSE. As for the performance of linear regression, a 0.587 RMSE places us within top 45 of the SemEval leaderboard.

VI. ii. KERNEL METHODS

We then trained one regression model based on kernel methods, Kernel Ridge Regression. Kernel ridge regression (KRR)[5] combines Ridge regression and classification (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function in the original space. So in some way, this model is very similar to a Support Vector Machine model. However, different loss functions are used: KRR uses squared error loss while support vector regression uses -insensitive loss, both combined with l2 regularization. After training and hyper parameter tuning, our best result from KRR was 0.596 RMSE.

VI. ii. SUPPORT VECTOR MACHINE

We tried SVR right after KRR. The model produced by support vector classification depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. The SVM searches for the maximized margin within the closest data points. So, SVM performs better when given a smaller dataset, since it needs to enumerate over all data points to search for the target margin. This SVR was the best SML model we implemented. With an RMSE of 0.577, it places us within top 40 f the SemEval leaderboard.

VII. DEEP LEARNING

Having studied DL approaches for NLP in class, we decided to build several DL architectures for our task. We hereby note that most of our ideas were inspired by [12], and the utilization of LSTM's to help us in our task came from [13]. We go through each one of the DL architectures in more details below. All embeddings mentioned in this section are 300D GloVe embeddings.

VII. i. FEEDFORWARD NEURAL NETWORK

Our baseline DL architecture is a simple FNN. The input to the FNN was the edited sentences. The FNN consisted of an embedding layer, 5 dense layers and a final output layer. ReLU was chosen as the activation function in the final layer because the range of output was 0-3 and the fact that ReLU can filter out negative activations. Through our experimentations, we found that, by using ReLU instead of linear activation on the last layer, the number of epochs before convergence became smaller (2-5 epochs fewer) and both the training and validation loss were lower (0.02-0.04 MSE lower). Overall, using ReLU meant that we did not have to update weights when the model produced negative results - so the models became easier easy to train. Our baseline FNN achieved a RMSE of 0.606. We note here that this baseline DL model is actually performing worse than the SML models.

VII. ii. 1-DIMENSIONAL CNN

We then employed 1-D CNN's for our task. The CNN take the edited sentences as input and passes them into an embedding layer. Afterwards the word vectors are passed into 5 *sets* of convolutions, one for each filter size from 2-6. Each *set* consisted of 128 feature maps followed by 32 feature maps followed by a global max pooling layer. The results of the 5 sets of convolutions were concatenated and then fed into two dense layers and then a final output layer which uses ReLU. This CNN architecture achieved 0.550 RMSE which places us within top 25 on the SemEval leaderboard.

VII. iii. ORIGINAL AND EDITED LSTM

We developed an LSTM architecture that not only considered the edited headlines but also the original headlines. The original and edited headlines are passed into embeddings and their word vectors are then separately fed into 2 layers of LSTM units. The LSTM outputs for each headline are then concatenated before being fed into 3 dense layers and a final output layer that uses ReLU. We had expected such an architecture to perform reasonably well, as it has information on the original headline from which the edited version is derived. This architecture met our expectation to a certain degree, as it achieved an RMSE of 0.573, which improves upon the baseline DL model and would place us within top 40 on the SemEval leaderboard.

VII. iv. BIDIRECTIONAL LSTM

We created a simple embedding-to-Bi-LSTM-to-dense-layer network that used the edited headlines as input and produced a grade for funniness. This network turned out to achieve around 0.560 RMSE, which puts us within top 30 of the SemEval leaderboard. We built this architecture because it is a stepping stone to the architecture in the next section and we had used the results from this architure to help us gain better insight into and tune the architecture in the next section.

VII. v. LSTM +Bi-LSTM

Building on top of the concept mentioned in the prior section, the idea behind this architecture is to first have a layer of LSTM units briefly process the input text, during which the finer points of the text are captured, and then have Bi-LSTM magnify and utilize those more refined information to make better predictions. The concept of this architecture is much like an encoder-decoder that shrinks the input down to its essence before expanding upon that and producing the output. This architecture is actually our best performing model in this project. The RMSE of this model on the testing set is 0.545 which places us within top 20 on the SemEval leaderboard.

VII. vi. BERT

We utilized a pre-trained BERT model from huggingface.co and built LSTM layers on top of it to adapt the BERT language model to our specific task. BERT, as a language model, is expected to understand more of the English context and thus perform better than the other models. This model took the longest to train and achieved only moderately good results. The RMSE after training was 0.574, which places us within top 35 on the SemEval leaderboard.

VIII. ENSEMBLING

We ensembled two of our SML models, namely our ridge regression model and our SVR, with our best DL architecture (VII. v) by taking the mean, minimum, and maximum of the outputs of the models. We found that ensembling the models in this way did not, in fact, improve the performance of our system. Ensembling by taking the mean achieved .568 RMSE, ensembling by taking the minimum achieved 0.554 RMSE, and ensembling by taking the 0.594 RMSE. The original performance of our best DL architecture was 0.545. Ensembling SML and DL, therefore, actually did not help our cause.

IX. TABLE OF RESULTS

We present here in this section a summary of our models and their RMSE on the testing set.

Model	RMSE
Linear Regression	0.587
Linear Regression (Ridge)	0.587
Linear Regression (Lasso)	0.594
Kernel Ridge Regression	0.596
SVR	0.577
FNN	0.606
1-D CNN	0.550
LSTM (original and edited)	0.573
Bi-LSTM	0.560
LSTM + Bi-LSTM	0.545
BERT	0.574

X. ANALYSIS AND DISCUSSION

We would like to make several observations in this section regarding our results and models. First we note that, with the exception of our baseline DL model, all other DL models reached around 0.58 RMSE after their first epoch. Many of them were able to improve their results after further training – but the improvements were very marginal. In fact, theshttps://discord.gg/A6zhUNe DL models began to overfit the training data within 10 epochs. More training after that no longer help the models generalize and will cause the models' performances on the testing set to decrease significantly. We actually tried other

loss functions like Mean Absolute Error, Log-cosh, and Huber loss, but using these loss functions did not help smooth the convergence of our models. Overall, we feel that these DL architectures have an astounding amount of parameters with which they can adjust and fit the data well. But perhaps they can do that *too well*¸ as in overfitting is a serious issue and early stopping or the restoration of the parameters that produced to the best output are very much necessary.

We also conject here about why our BERT model did not do better than LSTM's and Bi-LSTM's. We think that BERT, due to its nature, is better at handling longer sequences of texts. [14] All of our data, however, are edited headlines that are meant to be one-liner jokes. The longest sentence in our dataset contains only 23 words. We suspect that, for this reason, BERT did not achieve better results

We also want to acknowledge the relative competence of our SML models, especially considering their convenience. The SML models took very little effort to build and train, comparatively, and they produced fairly acceptable results. We also appreciate SML models' higher interpretability compared to the DL models. Since we can always tell what goes into each of the models and from there gather the factors that influence the output, in this case the characteristics of funnier headlines. These are benefits of SML models that are helpful to tasks not just in NLP but in any modeling-related field.

We are satisfied with the system we've built and its performance, given the time and resource constraints we had. Had our time and computational resources been ampler, we would have liked to try more models and do more hyper parameter tuning so that our models would achieve even better results.

XI. CONCLUSION

This project is meaningful to each one of us in our group. (We are all humorous people, although we may not show it during class.) As CS students taking NLP, we really enjoy the challenge of applying computational methodologies to assess humor. We have some important take-aways from this project. One of them is that, given proper feature extraction, SML models are very convenient to build and use for NLP tasks. In addition, they can produce some fairly competent results, even when compared to sophisticated DL models. We have also experienced first-hand how DL architectures are harder to train and interpret compared to SML models. We note that, when using a pre-trained language model like BERT, one should keep in mind of the domain of their corpus and how closely that domain relates to or resembles what BERT was meant to model. It is not always true that more complex models are better. Successfully utilizing empirical methods in NLP requires hard work and ingenuity in every step of the way – from understanding the data in exploratory data analysis to engineering and extracting features to building models and leveraging the complexity of models with the potential gain in performance. We believe building good NLP models take time, effort and just a dash of innovation.

XII. REFERENCES

[1] Hossain, N., Krumm, J., & Gamon, M. (2019). " President Vows to Cut< Taxes> Hair": Dataset and Analysis of Creative Text Editing for Humorous Headlines. *arXiv preprint arXiv:1906.00274*.

[2] Kannan, S., & Gurusamy, V. (2014). Preprocessing techniques for text mining. *International Journal of Computer Science & Communication Networks*, *5*(1), 7-16.

[3] Forman, G. (2008, October). BNS feature scaling: an improved representation over tf-idf for svm text classification. In *Proceedings of the 17th ACM conference on Information and knowledge management* (pp. 263-270).

[4] Mihalcea, R., & Pulman, S. (2007, February). Characterizing humour: An exploration of features in humorous texts. In *International Conference on Intelligent Text Processing and Computational Linguistics* (pp. 337-347). Springer, Berlin, Heidelberg.

[5] Murphy, K. P. *"Machine Learning: A Probabilistic Perspective"* - chapter 14.4.3, pp. 492-493, The MIT Press, 2012

[6] Rochmawati, D. (2017). Pragmatic and rhetorical strategies in the english-written jokes. *Indonesian Journal of Applied Linguistics*, *7*(1), 149-159.

[7] Shahaf, D., Horvitz, E., & Mankoff, R. (2015, August). Inside jokes: Identifying humorous cartoon captions. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1065-1074).

[8] Morreall, J. (2016). Philosophy of humor. In Edward N. Zalta, editor, *The Stanford Encyclopedia*

of Philosophy, winter 2016 edition. Metaphysics Research Lab, Stanford University.

[9] Tomoioagă, A. (2015). A linguistic analysis of jokes. *Discourse As A Form Of Multiculturalism In Literature And Communication, Section: Language And Discourse Arhipelag Xxi Press, TîrguMureș*.

[10] Binsted, K., Pain, H., & Ritchie, G. D. (1997). Children's evaluation of computer-generated punning riddles. *Pragmatics & Cognition*, *5*(2), 305-354.

[11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, 2825-2830.

[12] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

[13] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

[14] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.