Hitachi at SemEval-2020 Task 7: Stacking at Scale with Heterogeneous Language Models for Humor Recognition

Terufumi Morishita[*], **Gaku Morio**[*], **Hiroaki Ozaki** and **Toshinori Miyoshi**
Hitachi, Ltd.
Resarch and Development Group
Kokubunji, Tokyo, Japan
{terufumi.morishita.wp, gaku.morio.vn,
hiroaki.ozaki.yu, toshinori.miyoshi.pd}@hitachi.com

Abstract

This paper describes the winning system for SemEval-2020 task 7: *Assessing Humor in Edited News Headlines*. Our strategy is **Stacking at Scale (SaS)** with heterogeneous **pre-trained language models (PLMs)** such as BERT and GPT-2. SaS first performs fine-tuning on numbers of PLMs with various hyperparameters and then applies a powerful stacking ensemble on top of the fine-tuned PLMs. Our experimental results show that SaS outperforms a naive average ensemble, leveraging weaker PLMs as well as high-performing PLMs. Interestingly, the results show that SaS captured *non*-funny semantics. Consequently, the system was ranked 1st in all subtasks by significant margins compared with other systems.

1 Introduction

The recognition of humor in text has been receiving much attention (Barbieri and Saggion, 2014; Hossain et al., 2019). Accordingly, SemEval-2020 task 7, *Assessing Humor in Edited News Headlines* (Hossain et al., 2020a), which aims at automatically recognizing humor in hand-edited news headlines, was held with two subtasks: Subtask 1, which aims at predicting a funny score for an edited news headline, and Subtask 2, which aims at predicting the funnier headline of two given edited headlines.

In this paper, we pursue humor recognition with a large-scale stacking ensemble (hereafter **Stacking at Scale** or **SaS**), by leveraging **pre-trained language models (PLMs)**. SaS is based on an ensemble method where a meta-estimator is trained to predict labels from the outputs of base models, finding the best combinations of the base models (Wolpert, 1992). Hence, there are two steps in SaS: (i) fine-tuning numbers of heterogeneous PLMs, including BERT (Devlin et al., 2019), GPT-2 (Radford et al., 2019), RoBERTa (Liu et al., 2019), Transformer-XL (Dai et al., 2019), XLNet (Yang et al., 2019), and XLM (Lample and Conneau, 2019), with various hyperparameters, obtaining rich and diverse models, and (ii) training a meta-estimator on top of these PLMs.

Our experiments, fusing up to 1750 PLMs in total, indicate that SaS successfully leverages weaker PLMs as well as high-performing PLMs. Consequently, our system is ranked 1st on both subtasks with significant margins to others. Interestingly, analyses show that SaS learned (relatively) *non*-funny semantics while still struggling to understand the funniest semantics. To the best of our knowledge, this is the first experiment that involves thousands of diverse of PLMs, revealing the current strengths and limitations of PLMs in automatic humor recognition. We also provide useful insights obtained from rich analyses.

2 Background

Work related to humor recognition has been done in recent years (Khodak et al., 2018; Barbieri and Saggion, 2014; Reyes et al., 2012). Khodak et al. (2018) introduced a large-scale annotated corpus of sarcasm and provided baseline systems for sarcasm detection. Barbieri and Saggion (2014) widely

[*]Contributed equally.

Proceedings of the 14th International Workshop on Semantic Evaluation, pages 791–803
Barcelona, Spain (Online), December 12, 2020.

investigated features for automatically detecting irony and humor. SemEval-2020 task 7 (Hossain et al., 2020a) aims at automatically detecting humor in hand-edited news headlines and was introduced by (Hossain et al., 2019). We worked to solve the problem by utilizing a number of PLMs with stacking.

3 Task Formalization

As we described in the above, Subtask 1 aims at predicting a "funny score", a real-value in the range of $[0, 3]$ (0 = "Not", 1 = "Slightly", 2 = "Moderately", 3 = "Funny") for an edited headline. We formalized the task as a *sentence-pair regression*. Subtask 2 aims at predicting the funnier headline of two edited headlines originating from the same headline. We take an approach to utilizing the model of Subtask 1, that is, estimating the scores of the edited headlines and choosing the one having the higher score.

4 Fine-Tuning Pre-Trained Language Model (PLM) on Sentence-Pair Regression

Figure 1 shows an overview of our proposed model architecture. Given a pair of edited and original headlines, we apply PLM, BiLSTM layers, a dot-product attention layer, a pooling layer, and a feed-forward layer successively to predict funny scores.

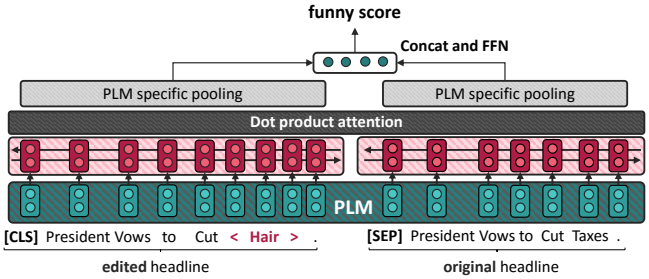

Figure 1: Overview of proposed model

Preprocessing: We concatenate two headlines. Tokenization is conducted by a PLM-specific tokenizer. We surround the edited tokens with two special marking tokens, "<" and ">." We insert special tokens (e.g., [CLS] and [SEP]) if necessary as required for each PLM. The implementations are described in detail in Section 6.1.

4.1 Intra- and Inter-Headline Encoding

To recognize inner-headline semantics, we first apply headline-wise multi-layered BiLSTM (Graves et al., 2013) as follows:

$$\mathbf{h}_i^{(\text{BiLSTM})} = \begin{cases} \text{BiLSTM}\left(\mathbf{h}_{\text{start\_edit}}^{(\text{PLM})}, \ldots, \mathbf{h}_{\text{end\_edit}}^{(\text{PLM})}\right)_i, & \text{if start\_edit} \leq i \leq \text{end\_edit} \\ \text{BiLSTM}\left(\mathbf{h}_{\text{start\_origin}}^{(\text{PLM})}, \ldots, \mathbf{h}_{\text{end\_origin}}^{(\text{PLM})}\right)_i, & \text{if start\_origin} \leq i \leq \text{end\_origin} \end{cases}$$

where $\mathbf{h}_i^{(\text{PLM})}$/$\mathbf{h}_i^{(\text{BiLSTM})}$ are the PLM/BiLSTM representation of the i-th token and (start_edit, end_edit)/(start_origin, end_origin) represent the starting/ending positions of the edited/original headlines.

Next, $\mathbf{h}_i^{(\text{BiLSTM})}$ are fed into the global dot-product-attention to capture inter-headline semantics, producing final hidden embeddings $\mathbf{h}_i$.

4.2 Funny Score Regression

We employ a headline-wise pooling layer and predict the funny score with a feed-forward network (FFN):

$$\hat{y} = \mathbf{v}^\top \text{FFN}\left(\text{POOLING}_{\text{PLM}}\left(\mathbf{h}_{\text{start\_edit:end\_edit}}\right) \oplus \text{POOLING}_{\text{PLM}}\left(\mathbf{h}_{\text{start\_origin:end\_origin}}\right)\right),$$

where $\oplus$ is a concatenation operation. $\text{POOLING}_{\text{PLM}}$ is a PLM-specific embedding pooling function. For example, for BERT, it takes the embeddings of the first tokens of two headlines ("[CLS]" and "[SEP]"). The details are in Table 7 of Appendix B. We trained the model with mean squared error loss.

5 Stacking at Scale

We further propose large-scaled ensemble, called **Stacking at Scale (SaS)**, based on a two-layer stacking ensemble (Wolpert, 1992), where the first-layer models (i.e., base models) are fine-tuned PLMs with different hyperparameter sets, and the second-layer model (i.e., meta-estimator) is another regression model. This may select the best combinations of the base models to produce more robust predictions.

Figure 2 shows a schematic view and the algorithm steps of SaS. The key attributes are (i) using heterogeneous PLMs for base models, (ii) generating diverse hyperparameter sets for the base models,

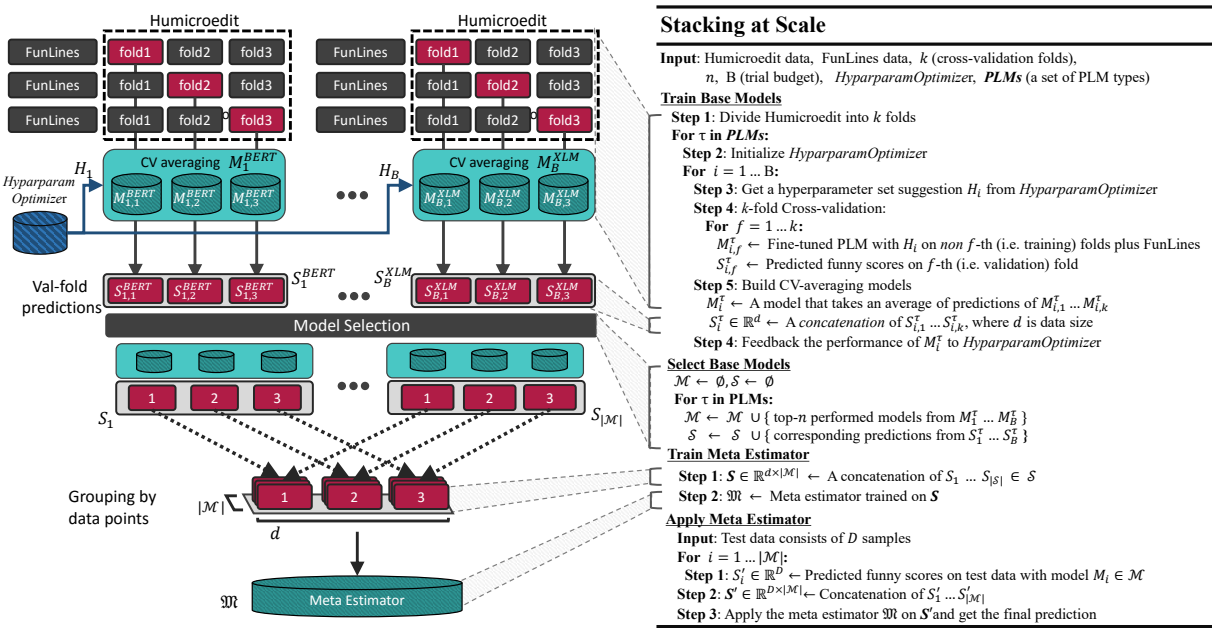

Figure 2: Simplified example of Stacking at Scale with 3-fold cross-validation

and (iii) performing cross-validation (CV) during the whole process. CV is used for accumulating *label leakage-free* prediction data over the whole training dataset used for the meta-estimator training as well as measuring the accurate performances of models to select better base models. Since SaS requires enormous computations, discussions on complexity are given in Appendix A.

5.1 Base Model Hyperparameter Generation

We pursue diversity for the base models by generating numbers of various hyperparameter sets. To generate sets with reasonable performances in a relatively small number of trials, we utilize a hyperparameter optimization framework. It seeks the best hyperparameter set by performing an iterative search, (i) suggesting (possibly better) sets given the sets already found and their performances and (ii) measuring the performances of the newly suggested sets (see **Train Base Models** in Figure 2). The performance for each set is measured on the basis of mean squared errors (MSEs) averaged over k validation folds of CV.

Since our purpose here is not only to find the best hyperparameter sets but to collect diverse sets with reasonable performances, we keep all the sets suggested during the search. After the search, we select the top performing n sets from each PLM type (see **Select Base Models** in Figure 2).

5.2 Meta-Estimator Training and Inference

Since the non-linearity laid on the dataset could have already been captured by PLMs, we use simple linear regression models for the meta-estimator. Suppose that the scores for a headline predicted by N base models are $\hat{y}_1, ..., \hat{y}_N$. The meta-estimator learns the weights w_i in the linear regression problem

$$\hat{y}_{\text{meta}} = w_0 + w_1\hat{y}_1 + ... + w_N\hat{y}_N \quad (1)$$

by using MSE loss with some regularization term. The input dimensionality N is (# of PLM types $\times$ n) because we pick the top n hyperparameter sets for each PLM type. For example, for **Train Meta Estimator** in Figure 2, $|\mathcal{M}| = |S| =$ (# of PLM types $\times$ n).

Overall, to predict funny scores with SaS, we (i) feed a headline pair into (# of PLM types $\times$ n $\times$ k) base models, obtaining (# of PLM types $\times$ n $\times$ k) predictions in total, (ii) take the CV-wise average of the predictions, reducing the dimension to (# of PLM types $\times$ n), and (iii) feed the CV-averaged predictions into the meta-estimator to get the final prediction (see **Apply Meta Estimator** in Figure 2).

6 Experiments

6.1 Settings

Offline Performance Measurements: Throughout our experiments, we estimated the performances of the models using the root mean squared error (RMSE) aggregated over the validation data of k=5 fold cross-validation[1] (hereafter **RMSE-CV**). Note that the aggregation is done over k=5 different sets of validation data, so we can measure the performances robustly.

Base Models: Table 1 shows the seven employed PLMs. We employed Optuna (Akiba et al., 2019) as the hyperparameter optimization framework. We generated 50 hyperparameter sets for each PLM. Therefore, in total, 350 models (= 7 types of PLMs $\times$ 50 sets of hyperparameters) or 1750 models including the CV variants ($\times$5 CV-folds) were built for the experiments. Details of on the hyperparameters are given in Appendix B. We tried many choices of n ranging from 1 to 50 to minimize the RMSE-CV.

Meta-Estimators: Two types of meta-estimators were employed: (i) Lasso regression (Tibshirani, 1996), i.e., linear regression with L1 regularization $\beta|w|$, and (ii) Ridge regression (Hoerl and Kennard, 1970), i.e., linear regression with L2 regularization $\beta||w||^2$. The strength parameter β was chosen from the default search values of scikit-learn (Pedregosa et al., 2011) to minimize the RMSE-CV.

Data: We used Humicroedit (Hossain et al., 2019) and FunLines (Hossain et al., 2020b), which are distributed officially. They have the same data format; however, they have slightly different label distributions (Hossain et al., 2020b). The official splits (i.e., train and dev) of Humicroedit are all concatenated to a single dataset, on which the cross-validation folds are built.

For training folds, we used both relevant datasets, Humicroedit (Hossain et al., 2019) and FunLines (Hossain et al., 2020b), to maximally capture funny semantics. However, for validation folds, we used only Humicroedit because the test data instances were taken only from Humicroedit and we wanted to measure the approximate model performances on the test set.

Implementation: We implemented the base models with jiant (Pruksachatkun et al., 2020), a transfer learning framework, which in turn utilizes Hugging Face's Transformers library (Wolf et al., 2019) for their implementation of PLMs, PLM-specific tokens (e.g., "[CLS]" and "[SEP]" for BERT), and a PLM-specific tokenizer. We implemented the meta-estimators using scikit-learn (Pedregosa et al., 2011). We employed the RidgeCV and LassoCV functions for Ridge/Lasso regressions. Both functions automatically find the best regularization strengths β.

Computational Resource: We employed up to 800 Volta (16-GB) GPUs offered by ABCI[2].

6.2 Results and Discussions

Official Ranking: We submitted the SaS-Ridge (n=20) system, i.e., SaS with the Ridge estimator using n=20 hyperparameter sets per PLM type, which performed the best in our pre-submission experiments. The model utilized 700 base models (=7 types of PLMs $\times$ 20 sets of hyperparameters $\times$ 5 CV-folds). The official ranking presented in Table 2 shows that our system is ranked 1st on both subtasks by significant margins to others. Hereafter, we analyze our system using Subtask 1 since we tuned our systems on it.

How Powerful is SaS?: We show ablation results for each PLM for $n = 1$ systems in Table 3. Most of the stacking models (shown as "SaS") performed better than single models ("single"), showing the effectiveness of fusing heterogeneous PLMs. Removing a PLM from SaS almost always degrades the performance regardless of the native performance of the removed model. This implies that not only the strongest PLMs but also the weaker PLMs are important for SaS. Hereafter, we use the single RoBERTa model as our baseline, which is the strongest model among the single models. Note that this baseline is competitive since it is with the best hyperparameter found in the 50-step hyperparameter optimization.

Figure 3 shows the change in performance for the total number of base models without CV variants (i.e., 7 types of PLMs $\times$ n). SaS-Ridge achieved its best performance around 100 models, and SaS-Lasso

[1] Let MSE_i be the MSE and n_i the number of instances for the ith-fold validation data. The aggregated RMSE is as follows.

$$\text{RMSE-CV} = \sqrt{\frac{1}{\sum_{i=1}^{k} n_i} \sum_{i=1}^{k} n_i \times \text{MSE}_i}$$

[2] **AI B**ridging **C**loud **I**nfrastructure provided by National Institute of Advanced Industrial Science and Technology (AIST).

PLM	model
BERT (Devlin et al., 2019)	large-uncased
GPT-2 (Radford et al., 2019)	medium / large
RoBERTa (Liu et al., 2019)	large
Transformer-XL (Dai et al., 2019)	wt103
XLNet (Yang et al., 2019)	large-cased
XLM (Lample and Conneau, 2019)	en-2048

Table 1: Seven PLMs employed in SaS. "model" represents specific pre-trained model variants of HuggingFace's Transformers library (Wolf et al., 2019)

.

Subtask 1		Subtask 2	
team	RMSE	team	accuracy
Hitachi (ours)	**0.49725**	**Hitachi (ours)**	**0.67428**
Amobee	0.50726	Amobee	0.66058
YNU-HPCC	0.51737	YNU-HPCC	0.65906
MLEngineer	0.51966	lmml	0.64688
lmml	0.52027	PALI	0.64460

Table 2: Official results for top five teams. Performances on official test data are shown.

model	RMSE-CV
SaS-Ridge (*n*=1)	**0.4998**
average ensemble	0.5071
SaS-Ridge (n=1) w/o BERT	0.5004
SaS-Ridge (n=1) w/o GPT-2 (M)	0.5003
SaS-Ridge (n=1) w/o GPT-2 (L)	0.5014
SaS-Ridge (n=1) w/o RoBERTa	0.5052
SaS-Ridge (n=1) w/o Transformer-XL	**0.4998**
SaS-Ridge (n=1) w/o XLNet	0.4999
SaS-Ridge (n=1) w/o XLM	0.5001
single BERT	0.5237
single GPT-2 (M)	0.5217
single GPT-2 (L)	0.5168
single RoBERTa	0.5109
single Transformer-XL	0.5565
single XLNet	0.5536
single XLM	0.5349

Table 3: Performance comparison of various models. RMSE aggregated over k=5 CV's validation data (RMSE-CV) is shown. **Top**: Ensemble models (SaS and average ensemble) over $n = 1$ base models from each PLM type. **Middle**: SaS models without a specific PLM. **Bottom**: single base models.

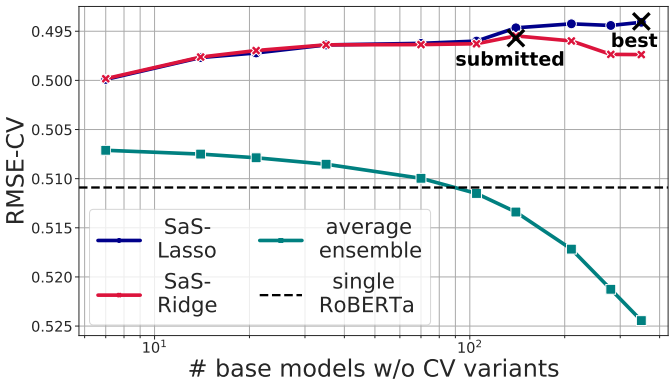

Figure 3: Change in performance for number of base models without CV variants. RMSE aggregated over k=5 CV's validation data (RMSE-CV) is shown. For comparison, average ensemble model and single RoBERTa model are also shown.

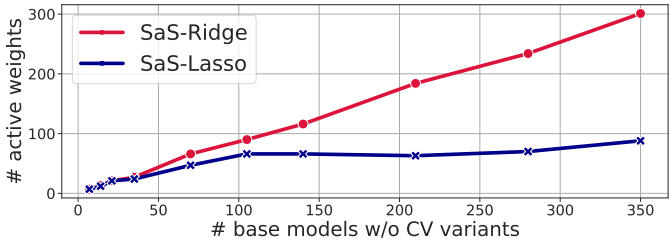

Figure 4: Change in number of active weights of SaS for number of base models without CV variants. Models were trained on our k=5 CV's training data. Values shown are averages over CV variant models.

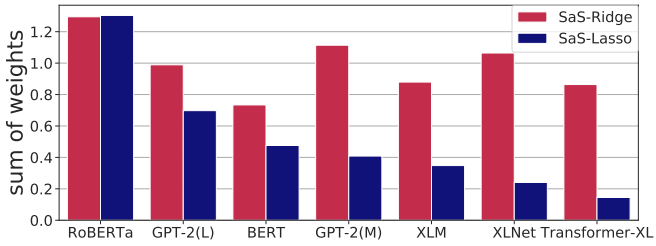

Figure 5: PLM-wise sum of absolute weights ($\sum_{i \in \text{PLM}} |w_i|$) of best SaS models, i.e., SaS-Lasso (n=50) and SaS-Ridge (n=20). Models were trained on our k=5 CV's training data. Values shown are averages over CV variant models.

kept its performance high over the stacking of 100 models, while the naive average ensemble got worse. This implies that at least nearly or over 100 PLMs are required to achieve the best performance for SaS. Also, SaS successfully utilized weaker models without harming the performance, while the naive average ensemble failed in that. To validate this, we plotted the numbers of active weights (i.e., the number of $w_i (i \geq 1)$ in eq. (1) that meet the condition $|w_i| \geq \text{threshold}(0.01)$) in Figure 4. Since Lasso is a *sparse* linear model, it constantly activated 80 to 100 PLMs, while Ridge's active weights increased linearly. The result indicates that utilizing the sparse model can automatically adjust the number of PLMs to be used.

Which Type of PLM is Useful?: We obtained contribution scores for each PLM type via the meta-estimator's weights. Figure 5 shows the PLM-wise sums of absolute weights; $\sum_{i \in \text{PLM}} |w_i|$, where

headline	gold	**SaS**	RoBERTa	reduction
U.S. ambassador to U.N. says Russia tearing down global ~~order~~ delivery	0.00	**0.63**	1.11	0.48
North Carolina Governor Says He 'll Issue Executive Order For Full LGBTQ ~~Rights~~ alphabet	1.40	**1.12**	0.62	0.50
Hillary Clinton : Democrats Who Are Pro-Life Must ~~Vote~~ strip to Promote Abortion	1.60	**1.38**	0.46	0.92
'I certainly meant no ~~disrespect~~ respect' : Kellyanne Conway addresses her pose in the Oval Office photo	2.40	**0.77**	0.74	0.03
Rocks falling into oceans , not climate change , causing sea levels to rise according to one ~~congressman~~ toddler.	2.60	**0.85**	0.82	0.03
Trump 's ' strategy ' on ~~Afghanistan~~ Presidency : Let the next president figure it out	2.60	**0.80**	0.99	-0.19
Hillary Clinton Supporters Filed a Complaint Against ~~Bernie Sanders~~ themselves - And Lost	3.00	**0.76**	0.91	-0.15

Table 4: Some sample headlines on which our best system, SaS-Lasso (n=50), reduced absolute errors by large margins (**top**) and by small (or sometimes negative) margins (**bottom**). Besides headlines, we show gold funny score ("gold"), prediction made by our system ("**SaS**"), with single RoBERTa ("RoBERTa"), and error reduction over baseline RoBERTa ("reduction").

w_i are the weights in eq. (1). RoBERTa and GPT-2 seemed to be the most preferable models, consistent with the results of excluding the models shown in Table 3 (shown as *w/o*). However, the plot also indicates that the stacking succeeded in leveraging weaker models as well as the best models.

What Did SaS Solve?: Figure 6 shows a sample distribution over the gold funny scores (**top**) and the mean absolute error (i.e., $e_{\text{SaS}} = |\hat{y}_{\text{meta}} - y_{\text{gold}}|$) and the mean absolute error reductions (i.e., $e_{\text{RoBERTa}} - e_{\text{SaS}}$) over the single RoBERTa baseline (**bottom**). SaS improved performance for not- to slightly-funny ([0-1.5]) headlines, while having similar or degraded performance for the funnier ([1.5-3.0]) headlines. In short, **SaS learned (relatively) *non*-funny semantics.** Since these headlines are the majority, SaS also gained overall performance improvements.

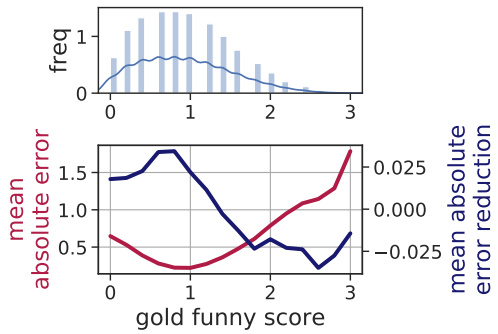

Figure 6: **Top**: Gold funny-score distribution. **Bottom**: Mean absolute error and mean absolute error reduction of our best model, SaS-Lasso (n=50), over single RoBERTa. Bin width was 0.2.

Case Study: Why Did SaS Learn Non-Funny Semantics? Table 4 shows sample headlines. The top rows show samples on which our best system, SaS-Lasso (n=50), reduced the errors over the single RoBERTa baseline by large margins. These headlines had small funny scores, and it seems that we can understand the non-funniness from the headline text itself without needing much external knowledge, and, in particular, some of the funniness comes only from the ***bizarreness*** or ***incongruity*** of the edited headlines. It is natural for PLMs to detect these types of non-funniness because they are trained on large amounts of corpora and could have learned to detect the unnaturalness of the given texts. We estimate that SaS enhanced this ability by combining the heterogeneous PLMs.

The bottom rows show headlines with small (or sometimes negative) error reductions. These headlines had large funny scores and seemed to be expressing ***irony***. Irony does not express intentions directly in text and rather relies on a reader's inference using sufficient common sense or background knowledge, especially on current topics. Given that SaS could have chosen the best combination of PLMs and that even SaS had no performance gain for such headlines, it is likely that such knowledge is not contained in any of the PLMs. This suggests the current limitation of PLMs on the humor recognition tasks.

7 Conclusion

In this paper, we proposed a top performing model for the task of humor recognition. We fused thousands of pre-trained language models by Stacking at Scale. Experimental results showed the incredible performance of the Stacking at Scale, and at the same time, also revealed the current limitation of pre-trained language models. For future work, we will explore injecting common sense or background knowledge into models to understand humor better.

Acknowledgments

Computational resource of AI Bridging Cloud Infrastructure (ABCI) provided by National Institute of Advanced Industrial Science and Technology (AIST) was used.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 2623–2631, New York, NY, USA. ACM.

Francesco Barbieri and Horacio Saggion. 2014. Automatic detection of irony and humour in twitter. In *ICCC*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy, July. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. 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)*, pages 4171–4186, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Alex. Graves, Abdel rahman. Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649.

A. E. Hoerl and R. W. Kennard. 1970. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67.

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)*, pages 133–142, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Nabil Hossain, John Krumm, Michael Gamon, and Henry Kautz. 2020a. Semeval-2020 Task 7: Assessing humor in edited news headlines. In *Proceedings of International Workshop on Semantic Evaluation (SemEval-2020)*, Barcelona, Spain.

Nabil Hossain, John Krumm, Tanvir Sajed, and Henry Kautz. 2020b. Stimulating creativity with FunLines: A case study of humor generation in headlines. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 256–262, Online, July. Association for Computational Linguistics.

Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. 2018. A large self-annotated corpus for sarcasm. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *Advances in Neural Information Processing Systems (NeurIPS)*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Yada Pruksachatkun, Phil Yeres, Haokun Liu, Jason Phang, Phu Mon Htut, Alex Wang, Ian Tenney, and Samuel R. Bowman. 2020. jiant: A software toolkit for research on general-purpose text understanding models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 109–117, Online, July. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Antonio Reyes, Paolo Rosso, and Davide Buscaldi. 2012. From humor recognition to irony detection: The figurative language of social media. *Data Knowledge Engineering*, 74:1–12, 04.

R. Tibshirani. 1996. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

David H. Wolpert. 1992. Stacked generalization. *Neural Networks*, 5:241–259.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5753–5763. Curran Associates, Inc.

A Time Complexity of Stacking at Scale

In this section, we discuss the time complexity of the Stacking at Scale (SaS) algorithm. We (i) first induce the theoretical time complexity of SaS and (ii) show measurements of the actual running time observed in our experiments, which is in accordance with those predicted by the theory.

The discussions are not that rigorous or exhaustive; however, we believe they are enough to offer readers rough estimations of the time complexity of SaS.

A.1 Theoretical Expressions

We estimate the time complexity of SaS, expressed by that of a single base-model system. The training phase complexity [eq. (2)] and the inference phase complexity [eq. (3)] are induced. In both cases, the dominant term comes from the base-model hyperparameter generation or inference. Thus, the SaS time complexity is (# of base models engaged in a phase) times larger than that of a single base-model system.

We first decompose the SaS algorithm into several steps and induce the time complexity of each step independently. Then, we aggregate the complexities to calculate the overall complexity of SaS.

A.1.1 Base-Model Hyperparameter Generation

Let $\tau^{\text{base}}_{\text{train}}(D_{\text{train}})$ be the time needed to train a single base model on the training data D_{train} with a specific setup (say, a specific PLM type, number of epochs, specific machine resource used, etc.). Let N_{train} be the number of base models (i.e., the number of unique hyperparameter sets) to be trained. The time complexity of base-model hyperparameter generation $T^{\text{base}}_{\text{train}}(D_{\text{train}})$ is estimated as follows.

$$T^{\text{base}}_{\text{train}}(D_{\text{train}}) = N_{\text{train}} \times \tau^{\text{base}}_{\text{train}}(D_{\text{train}})$$

Referring to Figure 2, N_{train} is expressed as:

$$N_{\text{train}} = P \times B \times k,$$

where P is the number of PLM types, B the hyperparameter-optimization step budget per PLM type, and k the number of cross-validation folds.

A.1.2 Base Model Inference

Let $\tau^{\text{base}}_{\text{infer}}(D_{\text{test}})$ be the time needed to execute inference with a single base model over the test data D_{test} with a specific setup. The time complexity of base model inference $T^{\text{base}}_{\text{infer}}(D_{\text{test}})$ is estimated as follows.

$$T^{\text{base}}_{\text{infer}}(D_{\text{test}}) = N_{\text{infer}} \times \tau^{\text{base}}_{\text{infer}}(D_{\text{test}})$$

Let $n(\leq B)$ the number of models per PLM type that are engaged in SaS. Then, N_{infer} is expressed as follows.

$$N_{\text{infer}} = P \times n \times k$$

A.1.3 Meta-Estimator Training

Since the inputs of the meta-estimators are the predictions over the training data D_{train} made by the base models, we must execute the inference of the base model over the training data D_{train} beforehand. Therefore, the time complexity of meta-estimator training $T^{\text{meta}}_{\text{train}}$ is expressed as:

$$T^{\text{meta}}_{\text{train}}(D_{\text{train}}) = T^{\text{base}}_{\text{infer}}(D_{\text{train}}) + \tau^{\text{meta}}_{\text{train}}(D_{\text{train}}),$$

where $\tau^{\text{meta}}_{\text{train}}(D_{\text{train}})$ is the time needed to train a meta-estimator with a specific setup.

A.1.4 Meta-Estimator Inference

The time complexity of meta-estimator inference $T^{\text{meta}}_{\text{infer}}$ is expressed as:

$$T^{\text{meta}}_{\text{infer}}(D_{\text{test}}) = T^{\text{base}}_{\text{infer}}(D_{\text{test}}) + \tau^{\text{meta}}_{\text{infer}}(D_{\text{test}}),$$

where $\tau^{\text{meta}}_{\text{infer}}(D_{\text{infer}})$ is the time needed to execute the inference of the meta-estimator for a specific setup.

A.1.5 Overall SaS Training

Overall, the time complexity of SaS training $T^{\text{SaS}}_{\text{train}}(D_{\text{train}})$ is as follows.

$$\begin{aligned} T^{\text{SaS}}_{\text{train}}(D_{\text{train}}) &= T^{\text{base}}_{\text{train}}(D_{\text{train}}) + T^{\text{meta}}_{\text{train}}(D_{\text{train}}) \\ &= PBk \times \tau^{\text{base}}_{\text{train}}(D_{\text{train}}) + Pnk \times \tau^{\text{base}}_{\text{infer}}(D_{\text{train}}) + \tau^{\text{meta}}_{\text{train}}(D_{\text{train}}) \\ &= PBk \times \tau^{\text{base}}_{\text{train}}(D_{\text{train}}) \times \left[1 + \frac{n}{B}\frac{\tau^{\text{base}}_{\text{infer}}(D_{\text{train}})}{\tau^{\text{base}}_{\text{train}}(D_{\text{train}})} + \frac{1}{N_{\text{train}}}\frac{\tau^{\text{meta}}_{\text{train}}(D_{\text{train}})}{\tau^{\text{base}}_{\text{train}}(D_{\text{train}})}\right] \end{aligned}$$

In many cases, the second term in the brackets is negligible given that $\frac{n}{B} \leq 1$ and that $\tau^{\text{base}}_{\text{infer}}(D_{\text{train}})/\tau^{\text{base}}_{\text{train}}(D_{\text{train}}) \ll 1$ often holds since, (i) in the training phase, we iterate over the dataset for several times, while, in the inference phase, we iterate only once, and, (ii) in the training phase, we need to back-propagate the gradients, while, in the inference, we do not. The third term can be negligible in the case where there are numbers of base models to train ($N_{\text{train}} \gg 1$) or the meta-estimator is "lighter" than the base models ($\tau^{\text{meta}}_{\text{train}}(D_{\text{train}})/\tau^{\text{base}}_{\text{train}}(D_{\text{train}}) \ll 1$; this indeed holds for our experiments since the base models are large neural networks, while the meta-estimators are just linear regressions. Thus, $T^{\text{SaS}}_{\text{train}}(D_{\text{train}})$ can be approximated only by the first term (i.e., base model training) as follows.

$$T^{\text{SaS}}_{\text{train}}(D_{\text{train}}) \sim PBk \times \tau^{\text{base}}_{\text{train}}(D_{\text{train}}) \tag{2}$$

Thus, the overall training complexity of SaS is *PBk* times larger than that of a base model.

A.1.6 Overall SaS Inference

The time complexity of SaS inference $T^{\text{SaS}}_{\text{infer}}(D_{\text{test}})$ is the same as that of the meta-estimator's inference $T^{\text{meta}}_{\text{infer}}(D_{\text{test}})$.

$$\begin{aligned} T^{\text{SaS}}_{\text{infer}}(D_{\text{test}}) &= T^{\text{base}}_{\text{infer}}(D_{\text{infer}}) + \tau^{\text{meta}}_{\text{infer}}(D_{\text{infer}}) \\ &= Pnk \times \tau^{\text{base}}_{\text{infer}}(D_{\text{test}}) + \tau^{\text{meta}}_{\text{infer}}(D_{\text{infer}}) \\ &= Pnk \times \tau^{\text{base}}_{\text{infer}}(D_{\text{test}}) \times \left[1 + \frac{1}{N_{\text{infer}}}\frac{\tau^{\text{meta}}_{\text{infer}}(D_{\text{test}})}{\tau^{\text{base}}_{\text{infer}}(D_{\text{test}})}\right] \end{aligned}$$

Again, the second term can be negligible in the case where there are numbers of base models engaged in SaS ($N_{\text{infer}} \gg 1$) or the meta-estimator is lighter than the base models ($\tau^{\text{meta}}_{\text{train}}(D_{\text{test}})/\tau^{\text{base}}_{\text{train}}(D_{\text{test}}) \ll 1$). Thus, $T^{\text{SaS}}_{\text{infer}}(D_{\text{test}})$ can be approximated only by the first term (i.e., base model inference) as follows.

$$T^{\text{SaS}}_{\text{infer}}(D_{\text{test}}) \sim Pnk \times \tau^{\text{base}}_{\text{infer}}(D_{\text{test}}) \tag{3}$$

Thus, the overall inference complexity of SaS is *Pnk* times larger than that of a base model.

A.2 Measurements of Running Times

In this section, we show the observed running times of the SaS algorithm. Please note that the results are not rigorous or exhaustive. The purpose here is rather to offer readers a taste of the order estimations of the computational time and resources needed to reproduce the SaS experiments.

A.2.1 Measurement of τ

We reposit the setting of the experiments in Table 5. For computational resources, we trained our base models using Volta (16-GB) GPUs (single model per single GPU). Some large models [i.e., GPT-2 (L) and XLM] were trained on Volta (32-GB) GPUs.

On average, the training time seemed to be about 30 minutes, that is:

$$\tau^{\text{base}}_{\text{train}}(D_{\text{train}}) \sim 0.5 \text{ hours},$$

Note that this estimation is really rough since $\tau^{\text{base}}_{\text{train}}(D_{\text{train}})$ depends on many factors including the PLM type, number of training epochs (mostly 8 or 16), batch size (1 to 16), and that the above value is only the average (or "marginal") actual running times.

With the same setting as the training, the inference time was observed to be:

$$\tau^{\text{base}}_{\text{infer}}(D_{\text{test}}) \sim 20 \text{ secs},$$

which is much smaller than th training time, as expected.

parameter	value	
P	7	Seven types of PLM are used. (see Table 1)
B	50	50 hyperparameter sets per PLM types are generated.
k	5	5-fold cross-validation is employed.
n	50	Our best system, Lasso($n = 50$) uses 50 base models per PLM type.
D_{train}	Humicroedit + FunLines	The concatenation is used. About 17k instances.
D_{test}	Official test data	About 3k instances.

Table 5: Setup of our experiments

A.2.2 Measurement of T

In our setting, the number of models trained (N_{train}) was as follows.

$$N_{\text{train}} = P \times B \times k = 7 \times 50 \times 5 = 1750$$

Thus, the total training time could be estimated theoretically as follows.

$$T^{\text{base}}_{\text{train}}(D_{\text{train}}) = N_{\text{train}} \times \tau^{\text{base}}_{\text{train}}(D_{\text{train}}) \sim 1750 \times 0.5 \text{ hours} = 875 \text{ hours}$$

As we observed, with 200 Volta GPUs, it took ~ 5 hours to train the whole SaS model, which is of the same order as the theoretically predicted training time.

The number of models engaged in the ensemble (N_{infer}) was estimated as follows.

$$N_{\text{infer}} = P \times n \times k = 7 \times 50 \times 5 = 1750$$

We have not measured the total inference time since the inference was executed at the same time as the training with our implementation. Therefore, we show only the theoretically expected inference time.

$$T^{\text{base}}_{\text{train}}(D_{\text{train}}) = N_{\text{train}} \times \tau^{\text{base}}_{\text{train}}(D_{\text{train}}) \sim 1750 \times 20 \text{ secs} \sim 10 \text{ hours}$$

B Base-Model Hyperparameter Generation

In this section, we describe the setup in detail and the results of the base-model hyperparameter generation.

B.1 Setup

We generated hyperparameter sets using Optuna (Akiba et al., 2019), a hyperparameter optimization framework. We used version 1.10. We started each optimization process using Optuna's default seed. For each PLM type, we generated 50 hyperparameter sets. At each step of the optimization process, we tried 5 hyperparameter sets in parallel. Therefore, in total, 10 steps were needed to try 50 hyperparameter sets.

Table 6 and Table 7 show the specific hyperparameter setups. Table 6 shows the hyperparameters and their (i) search range, (ii) the initial values, and (iii) the Optuna sampling functions used. Table 7 shows other PLM-specific, fixed hyperparameters.

name	explanation	search range	initial value	sampling function
learning rate	The global learning rate.	[1e-6, 3e-4]	3e-5	log-uniform
optimizer	The optimizer. "bert_adam" is the optimizer (adam with warmup) used in (Devlin et al., 2019)	"adam" or "bert_adam"	"adam"	categorical
gradient clipping	The value of gradient L2-norm clipping	-	5.0 (fixed)	-
max epochs	The max trainin epochs.	4, 8 or 16	8	categorical
early stopping patience	The patience of early stopping. The validation check is done with the intervals of 200 gradient steps.	-	5 (fixed)	-
BiLSTM layers	The number of the BiLSTM layers mentioned in section 4.1	1 or 2	2	categorical
BiLSTM hidden dim	The number of the hidden units of the BiLSTM layers.	256, 512 or 1024	512	categorical
attention hidden dim	The number of the hidden units of the dot-product-attention mentioned in section 4.2	256, 512 or 1024	512	categorical
BiLSTM and attention dropout	The dropout ratio of the BiLSTM layers and dot-product-attention.	[0.00, 0.30]	0.15	uniform
FFN hidden dim	The number of hidden units of the feed-forward layer mentiond in section 4.2.	256, 512 or 1024	512	categorical
FFN dropout	The dropout ratio of the feed-fowrard layer.	[0.00, 0.30]	0.20	uniform
CV fold (k)	The number of folds in the cross-validation.	-	5 (fixed)	-
activation function	The activation functions.	-	tanh (FFN) sigmoid (BiLSTM)	-

Table 6: Setup of each hyperparameter. Search range, initial value, and Optuna sampling function used are shown. Note that some hyperparameters are not searched for but fixed.

PLM	type	batch size	pooling
BERT (Devlin et al., 2019)	large-uncased	1	first token
GPT-2 (Radford et al., 2019)	medium / large	16 / 2	last token
RoBERTa (Liu et al., 2019)	large	16	average
Transformer-XL (Dai et al., 2019)	wt103	4	average
XLNet (Yang et al., 2019)	large-cased	16	last token
XLM (Lample and Conneau, 2019)	en-2048	1	average

Table 7: PLM-specific fixed hyperparameters. Batch size and embedding pooling function mentioned in Section 4.2 are shown. "First token" takes first token embedding from headline, "last token" takes last token embedding, and "average" takes average of all token embeddings in headline.

B.2 Results - The Best Values -

For the readers' convenience, we show the best hyperparameters found in our search in Table 8. Note that readers can reproduce these results by executing the hyperparameter search with the experimental setup described in the previous section.

	BERT	GPT-2 (L)	GPT-2 (M)	RoBERTa	Transformer-XL	XLNet	XLM
learning rate	9.06e-06	3.00e-05	4.50e-05	1.51e-05	4.38e-05	4.01e-06	7.02e-06
optimizer	adam	bert_adam	adam	bert_adam	adam	adam	adam
max epochs	8	8	16	16	8	16	8
BiLSTM layers	2	2	1	2	2	2	2
BiLSTM hidden dim	1024	512	512	256	1024	1024	512
attention hidden dim	1024	512	256	512	512	256	256
BiLSTM and attention dropout	1.59e-02	3.53e-02	7.99e-02	1.82e-01	1.16e-01	2.70e-01	1.52e-01
FFN hidden dim	512	256	1024	512	512	1024	512
FFN dropout	7.01e-02	1.56e-01	7.74e-02	7.49e-02	1.68e-01	2.82e-02	8.87e-02

Table 8: Best hyperparameters found in hyperparameter search. Performance was measured by RMSE-CV.