

Insert your title here

Bobak Farzin¹, Nombre Apellidos²

¹bfarzin@gmail.com

²Universidad o lugar de trabajo

Received: 15 June 2019 / Accepted: date

Abstract Our entry into the *HAHA 2019 Challenge* came in 3rd in the classification task and 2nd in the regression task. We describe our system and innovations and compare our results to a Naive Bayes classifier. All the code for our project is included in a GitHub¹ repository for easy reference and replication by others.

Keywords First keyword · Second keyword · More

1 Introduction

- ¡Socorro, me ha picado una vbora!
- ¿Cobra?
- No, gratis.²

Google Translation:

- Help, I was bitten by a snake!
- Does it charge?
- Not free.

Humor does not translate well because it often relies on double-meaning or a subtle play on the words or context. Thus, one would imagine building a system to classify the humor of tweets would be difficult. However, with transfer-learning and the Fastai library³, you can quickly get a high quality classifier working in a language you don't even speak which will easily outpace a Naive Bayes SVM baseline ⁴

¹ https://github.com/bfarzin/haha_2019_final

² <https://www.fluentin3months.com/spanish-jokes/>

³ <https://docs.fast.ai/>

⁴ <https://twitter.com/radamihalcea/status/1119020144389443585>

Rather than hand-crafted language features, we have taken an "end to end" approach building from the raw text to a final model that achieve the tasks as presented. Our paper lays out the details of the system and our code can be found in a GitHub repo for use by other researchers to extend the state of the art in sentiment analysis.

Contribution Our key contributions are three fold. First, we apply transfer-learning of a lanugage model based on a larger corpus of tweets. Second, we use a label smoothed loss to allow full training of the final model without gradual unfreezing. Third, we select the best classifier or regression model based on cross-validation and 20 random-seed initialization in the final network training step.

2 Task and Dataset Description

The *Humor Analysis based on Humor Annotation (HAHA) 2019* [5] competition asked for analysis of two tasks in the Spanish language based on a corpus of publicly collected data [3]:

- **Task1: Humor Detection:** Detemine if a tweet is humorous. System ranking is based on F1 score which will balance precision and accuracy.
- **Task2: Funniness Score:** If humorous, what is the average humor rating of the tweet. System ranking is based on RMSE.

The HAHA dataset includes labeled data for 24,000 tweets and a test set of 6,000 tweets (80%/20% train/test split.) Each record includes the raw tweet text (including accents and emoticons), a binary humor label, the number of votes for each of five star ratings and a "Funniness Score" that is the average of the 1 to 5 star votes cast. Examples and data can be found on the CodaLab competition webpage⁵.

3 System Description

We generally follow the method of ULMFiT [6] including pre-training and differential learning rates.

1. Train a language model (LM) on a large corpus of data
2. Fine-tune the LM based on the target task language data
3. Replace the final layer of the LM with a softmax or linear output layer and then fine-tune on the particular task at hand (classification or regression)

Below we will give more detail on each step and the parameters used to generate our system.

⁵ <http://competitions.codalab.org/competitions/22194/>

3.1 Data, Cleanup & Tokenization

3.2 Additional Data

For our initial training, we collected 475,143 tweets in the Spanish language using tweepy [1]. The frequency of terms, punctuation and vocabulary can be quite different from the standard Wikipedia corpus that is often used to train an LM.

In the fine-tuning step, we combined the labeled and un-labeled text data for the LM training.

3.3 Cleaning

We applied a list of default cleanup functions included in Fastai[] and added an additional one for this Twitter dataset.

- Add spaces between special chars (ie. !!! to ! ! !)
- Remove useless spaces (remove more than 2 spaces in sequence)
- Replace repetition at the character level (ie. **grrrreat** becomes **g xxrep r 3 eat**)
- Replace repetition at the word level (similar to above)
- Deal with ALL CAPS words replacing with a token and converting to lower case.
- **Addition:** Move all text onto a single line by replacing new-lines inside a tweet with a reserved word (ie. \n to **xxnl**)

The following example shows the application of this data cleaning to a single tweet:

```
Saber, entender y estar convencidos que la frase \
#LaESILaDefendemosEntreTodes es nuestra linea es nuestro eje.\
#AlertaESI!!!!
Vamos por mas!!! e invitamos a todas aquellas personas que quieran \
se parte.

xxbos saber , entender y estar convencidos que la frase \
# laesiladefendemosentretodes es nuestra linea es nuestro eje.\
xxnl # alertaesi xxrep 4 ! xxnl vamos por mas ! ! ! e invitamos a \
todas aquellas personas que quieran se parte.
```

3.4 Tokenization

We used sentencepiece [7] to parse into sub-word units and reduce the possible out-of-vocabulary (OOV) terms in the data set. We selected a vocab size of 30,000 and used the byte-pair encoding (bpe) model.

4 Training and Results

4.1 LM Training and Fine-tuning

We train the LM using a 90/10 training/validation split and report the validation loss and accuracy of next-word prediction on the validation set. For the LM, we selected a AWD-LSTM [8] model included in Fastai with QRNN[2] units, 2304 hidden-states, 3 layers and a softmax layer on the end to predict the next-word. We tied the embedding weights on the encoder and decoder for training. We performed some simple tests with LSTM units and with a Transformer Language model and found all the models were similar in performance during LM training. We chose the QRNN units to speed up training. The model has about 60 million trainable parameters.

Our loss is label smoothing[9] of the flattened cross-entropy loss. Parameters used for training and finetuning are shown in Table 1. For all the networks we applied a dropout multiplier that scales the dropout used throughout the network. We used the Adam optimizer with weight decay as indicated in the table.

Following the work of Smith[10] we found the largest learning-rate that we could apply and then ran a one-cycle policy for a single epoch. We then ran subsequent training with one-cycle and lower learning rates indicated in Table 1.

Table 1 LM Training Parameters

Param	LM	Fine-Tune LM
Weight Decay	0.1	0.1
Dropout Mult	1.0	1.0
Learning Rate	1 epoch at $5 * 10^{-3}$	5 epochs at $3 * 10^{-3}$
Cont. Training	15 epochs at $1 * 10^{-3}$	10 epochs at $1 * 10^{-4}$

4.2 Classification and Regression Fitting

Again, following the play-book from [6], we change out the head of the network to a softmax or linear output layer and then load the LM weights for the layers below. We train just the new head from the random initialization, then unfreeze the entire network and train with differential learning rates.

With the same learning rate and weight decay we apply a 5-fold cross-validation on the outputs and take the mean across the folds as our ensemble. We sample 20 random seeds (see more in section 4.3) to find the best initialization for our gradient descent search. From these samples, we select the best validation F1 metric or MSE for use in our test submission.

4.2.1 Classifier setup

For the classifier, we have a softmax head and label smoothing loss which allows us to train without gradual unfreezing. We oversample the minority class to balance the outcomes for better training using SMOTE [4].

4.2.2 Regression setup

For the regression, we fill all #N/A labels with scores of 0. We add a linear head output and mean-squared-error (MSE) loss function.

Table 2 Classification and Regression Training Parameters

Param	Value
Weight Decay	0.1
Dropout Mult	0.7
Learning Rate (Head)	2 epochs at $1 * 10^{-2}$
Cont. Training	15 epochs with diff lr: $(1 * 10^{-3} / (2.6^4), 5 * 10^{-3})$

4.3 Random Seed as a Hyperparameter

For classification and regression, the random seed sets the initial random weights of the head layer. This initialization effects the final F1 metric you can achieve with your model.

Across each of the 20 random seeds, we average the 5-folds and get a single F1 metric on the validation set. The histogram of 20-seed outcomes is shown in Figure 1 and covers a range from 0.820 to 0.825 in the validation set. We selected our single best random seed for the test submission. With more exploration, a better seed could likely be found. We only tried a single seed for the LM training but one could do a similar search with random seeds there and select the best down-stream seed similar to [?]

4.4 Results

Below in Table 3 we layout three results from our submissions in the competition. The first is the Naive Bays SVM solution showing an F1 of 0.7548. Second is our first random seed selected for the classifier which produces a 0.8083 result. While better than the NBSVM solution, we can do better by picking the best F1 from the 20 seeds that we selected. That result is in the final row and gets our final submission value of 0.8099. Note that in our validation set achieved an F1 of 0.8254 but this is somewhat overfit from the many trials we have run and our final test set is a drop of 0.0155

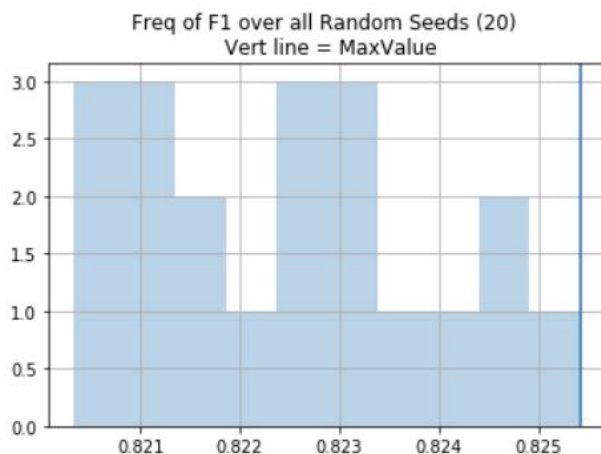


Fig. 1 Histogram of F1 metric averaged across 5-fold metric

Table 3 Comparative Results

	Accuracy	Precision	Recall	F1
NBSVM	0.8223	0.8180	0.7007	0.7548
First Seed	0.8461	0.7869	0.8309	0.8083
Best Seed	0.8458	0.7806	0.8416	0.8099

5 Conclusion

This paper describes our implementation of a neural net model for classification and regression in the HAHA 2019 challenge. Our solution placed third in Task 1 and second in task 2 in the final standings. We describe the data collection, pre-training and final model building steps. Additionally, we have open-sourced all the code used in this contest so it can be easily duplicate to further research in this task in the future.

Acknowledgements If you'd like to thank anyone, place your comments here and remove the percent signs.

References

1. Tweepy: Twitter for python! (2019). URL <http://github.com/tweepy/tweepy>
2. Bradbury, J., Merity, S., Xiong, C., Socher, R.: Quasi-recurrent neural networks. CoRR **abs/1611.01576** (2016). URL <http://arxiv.org/abs/1611.01576>
3. Castro, S., Chiruzzo, L., Rosá, A., Garat, D., Moncecchi, G.: A crowd-annotated spanish corpus for humor analysis. In: Proceedings of the Sixth International Workshop on Natural Language Processing for Social Media, pp. 7–11 (2018)
4. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: Synthetic minority over-sampling technique. J. Artif. Int. Res. **16**(1), 321–357 (2002). URL <http://dl.acm.org/citation.cfm?id=1622407.1622416>

5. Chiruzzo, L., Castro, S., Etcheverry, M., Garat, D., Prada, J.J., Rosá, A.: Overview of HABA at IberLEF 2019: Humor Analysis based on Human Annotation. In: Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2019), CEUR Workshop Proceedings. CEUR-WS, Bilbao, Spain (2019)
6. Howard, J., Ruder, S.: Fine-tuned language models for text classification. CoRR **abs/1801.06146** (2018). URL <http://arxiv.org/abs/1801.06146>
7. Kudo, T., Richardson, J.: Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. CoRR **abs/1808.06226** (2018). URL <http://arxiv.org/abs/1808.06226>
8. Merity, S., Keskar, N.S., Socher, R.: Regularizing and optimizing LSTM language models. CoRR **abs/1708.02182** (2017). URL <http://arxiv.org/abs/1708.02182>
9. Pereyra, G., Tucker, G., Chorowski, J., Kaiser, L., Hinton, G.E.: Regularizing neural networks by penalizing confident output distributions. CoRR **abs/1701.06548** (2017). URL <http://arxiv.org/abs/1701.06548>
10. Smith, L.N.: A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. CoRR **abs/1803.09820** (2018). URL <http://arxiv.org/abs/1803.09820>