Sabrina Jesmin & Rahul Damineni

AUTOMATED AUTHOR IDENTIFIER

1. PROBLEM

Cyber security companies predict potential attacks on their clients by monitoring deep and dark web forums where actors conspire and discuss potential bugs and vulnerabilities of clients that could be exploited to attack them. There are existing solutions that analyze texts and state if it's a threat purely based on content. These solutions tend to trigger a lot of false alarms because the discussion content is indiscernible from actual threat. The key indicator of authenticity of a discussion being a threat is its author. There are red hot malicious actors who are notorious for attacks in the past and if we see a threat from them, there's a high chance that it would lead to a potential attack.

The catch is, the malicious actors cover up their digital signatures effectively – making it impossible to attribute an anonymous threat to a known actor. However, the most intrinsic feature of such anonymous content is, the author's writing style.

Given a collection of documents, how to cluster the documents that are possibly written by the same author?

2. RELATED WORK

Historically, stylometry is a popular technique used to identify authors of anonymous content – when such content is written by famous people. The technique based on counting frequencies of word lengths called Mendenhall's Characteristic Curves of Composition (MCCC) is used for this. Another technique called Burrow's Delta which uses Term Frequency – Inverse Document Frequency (TF-IDF) on function and stop words to summarize the document into a vector and then use cosine similarity to find closest matching document.

3. SOLUTION

1. *We used Burrow's Delta technique (source code) to fingerprint each incoming document and tag it with a matching author. A curated set of function words have been used to construct a count vector of length 632. This technique failed and was able to perform only slightly better than random guess.*
2. *The other option is to use a neural network that can automatically derive these latent features. Care should be taken to setup this problem correctly as there's a chance of memorizing specific tokens*

from data instead of actually deriving general features that are representative of writing styles. ***Our best implementation correctly classified 85% of input queries from a balanced dataset.***

4. DATA

We used user comments from **Reddit** discussion forum as a dataset for both solutions. Comment dumps from May & September 2006 were processed to form an initial dataset of 73 authors posting at least 100 comments each on several subreddits. These 100 comments from each author are unique and are split into 60, 20, and 20 shares to make up train, eval and test splits respectively. The task is to learn a model using the train split and use it to map comments in dev and test split correctly. This dataset was directly used in Burrow's Delta technique and also initial version of neural network models.

To simplify learning, this dataset was processed further to form another one. Some comments in original dataset contain URLs which do not reveal anything about writing styles. Also, some comments are too short and general to be attributed to any author (examples: "I agree", "Alright!" etc.). This second dataset was formed by removing URLs and excluding comments that has less than 200 characters and other irregularities like non-English content. Please check this Jupyter notebook for more details. The new dataset only contains 24 authors and 100 comments for each.

5. IMPLEMENTATION DETAILS

Problem Setup & Model:

i. A gated recurrent neural network was built to take a sequence of tokens as input and converge it to probabilities of two classes: "match" or "no-match".

ii. **RedditComments** (source): The above data splits were transformed into a PyTorch dataset that would supply input-target pairs for all our neural networks. The following the details:

1. *Text input is tokenized using "spacy" tokenizer*
2. *For the binary classification problem, two random texts from possibly same or different author were concatenated with a "<SEP>" token between them. If the inputs are sampled from same author's comments, then the label would be "match".*
3. *The "mixing_strategy" argument is used to control logical form sparsity of the output distribution. If "ordered strategy" is used, the output samples will have more variation and help model converge faster.*
4. *The "p2nr" argument is used to regulate positives ("match") to negatives ratio of the output distribution.*
5. *Finally, "num_samples" was used to restrict the size of dataset to experiment faster.*

iii. The model's hyper-parameters are number of hidden gated recurrent units (`HIDDEN_REPR_DIM`), number of stacks of GRUs connected end-to-end (`NUM_HIDDEN_LAYERS`) and `LEARNING_RATE`. Binary Cross Entropy was used as the loss criteria. For a full list of parameters, please refer to source.

Experimentation & Fine-tuning:

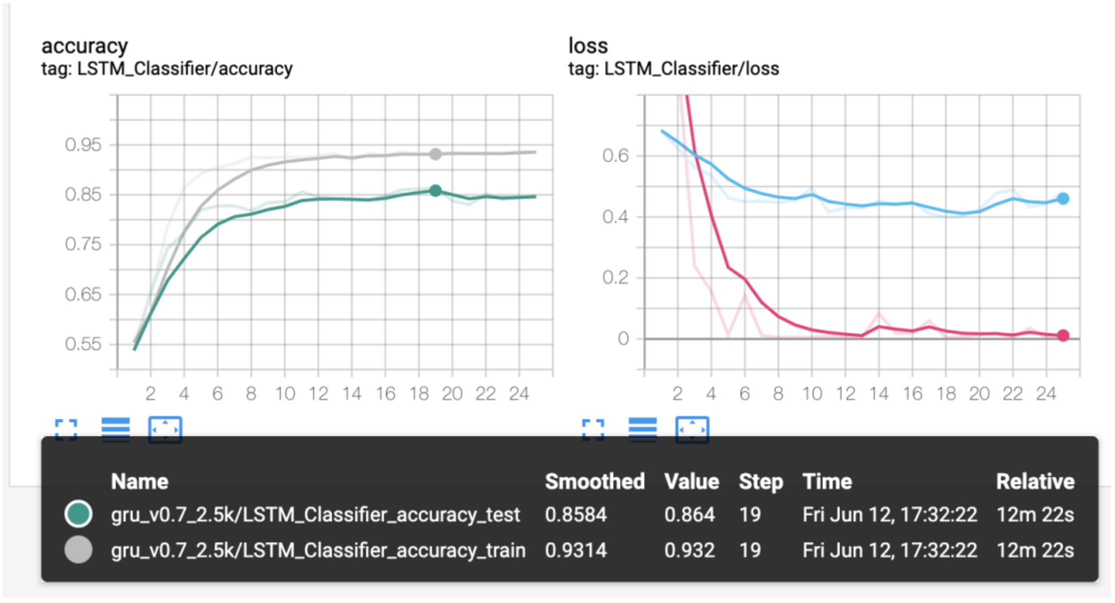
iv. All experiments are numbered and indexed based on the hyperparameters being used (logs).

v. First version of dataset with 73 authors didn't converge at all. After verifying that the training pipeline was setup right by performing over-fitting test, I revisited dataset and made the 24-author dataset by cleaning it. The reason for including longer lines only is to expose the encoder with more contents to recognize patterns.

vi. Even this dataset was taking too long to train and wasn't converging. At this point, I wasn't sure if the writing styles could be distinguished by this training pipeline at all. So, to verify this, I came up with the simplest dataset by using all qualified comments (~250) from top two authors in the original dataset. The rationale is, even a simpler model with less parameters should be able to identify the possibly stark differences in two authors' writing styles (if at all). I also used "ordered mixing strategy" to increase the variance in the data distribution seen by the model. The model started to converge and the validation set (whose examples are from the same author but never used for training) accuracy came out as **~62%.**

vii. Although, the input comments are concatenated from different subreddits (which are contextually quite different), I was worried that that 62% may have been because of memorized rare words and not because of the model actually learning discriminative features. So, I setup another experiment to train the same model using 24-authors' comments. If it were actually memorizing rare words, the accuracy shouldn't improve – as the distribution of rare-words across the comments should relatively be same. But it is actually learning patterns in the data, this extra surge of data should improve dev accuracy by a good amount: came out to be **~71%**

viii. A little more fine-tuning resulted in the personal best accuracy of **~85%**



	Name	Smoothed	Value	Step	Time	Relative
●	gru_v0.7_2.5k/LSTM_Classifier_accuracy_test	0.8584	0.864	19	Fri Jun 12, 17:32:22	12m 22s
●	gru_v0.7_2.5k/LSTM_Classifier_accuracy_train	0.9314	0.932	19	Fri Jun 12, 17:32:22	12m 22s

Inferencing for DBMS:

ix. Once the model is trained, it is expected to be general. (**haven't tested this!**) In the sense, it should work if we want to identify if two texts belong to some unseen author. Even though there's a covariate shift, this should work because the original 24 authors are selected randomly, and the training data is fairly cross-domain.

x. Based on this assumption, the inferencing script (source) would be given a trained model and a collection of users with textual content as features. The inferencing script learns the mean of each user's writing style representation (the input encoding of the user's text as seen from encoder stack output).

xi. It would then use these representations and to tag an incoming document by:

1. *preprocessing it for inference on the model*
2. *Retrieve its representation by passing it through the trained model*
3. *Comparing the representation with existing users' mean representations to find a match or tag it as a new author.*

6. SHORT FALLS & FUTURE WORK

A. We don't know what the model is learning: Inspecting the saliency maps may reveal if the procedure actually worked.

B. We don't know if the 85% validation accuracy could scale during inferencing: No empirical studies were performed as there was a difficulty porting trained model from Pelican to local. So, we don't really know if the assumption made in [ix] holds.

C. 85% isn't really great for classification task: that would mean, even among the author's writing styles the model already saw, it misclassified 15% of samples.

  i. The future work could involve fine-tuning the existing model.
  ii. Using transfer learning boost via T5 or BERT
  iii. Fine-tune BERT/ T5 on natural conversations ONLY so it may help catch the discernable features in writing styles.

Open-sourced code:
https://github.com/RahulDamineni/anonymous_author_matcher/

7. REFERENCES:

A. Using NLP to Identify Redditors Who Control Multiple Accounts