fake-real-news: A News Classifier in the Fight against Disinformation
Interim Prototype Report

Huang, Sheng 3035534103 & Li, Yik Wai 3035566015

Objectives - Revisited

- To build and train a classification model for the fair and impartial judgement on the nature and authenticity of a news article.
- To integrate that model with a web framework and make it an easy-to-use web app.
- To provide easy method for model update in the backend given the availability of newer and/or better datasets and user feedback.

The news classifier is a text classifier, which categorizes texts into organized groups. The major differences between a news classifier and other text classifiers lie in its implications.

The classifier cannot visit the places where events happened; it cannot interview people involved in the stories; it cannot know the intention of the publishers when they put out the story. Yet, it tries to inform people of the nature and authenticity of a news article. If done wrong, its work could undermine the integrity of true journalism, or further the cause of bad actors or malign forces with an agenda.

This contributes to the difficulty in the fight against disinformation. If we become too conservative, we will lose ground in this fight as disinformation spreads like fire. But if we become too assertive, we will slide into a dystopian world that has institutions equivalent to the Ministry of Truth from George Orwell's *Nineteen Eighty-Four*.

We do not aim to be the arbiter of truth, nor can we replace professional fact checkers, who can access so much more resources in the real world other than the news piece itself.

However, there are linguistic patterns we could follow in the news article itself, and with enough data, we could use those patterns to categorize the news pieces.

To further help the users understand the nature of the news articles they are reading, we do not just categorize articles into real and fake, because not all the reporting could be interpreted simply as such. For example, some articles are heavy on political opinions. We cannot dictate the realness or fakeness of a political opinion, and hence we would put it in a category called "political".

Prototype

For the purpose of building a prototype, we used only a small portion (~ 1/459) of the whole dataset we have, to test the effectiveness of our initial idea. Also, to simplify things for the prototype stage, we only used the most important `content` and the `type` columns of the dataset, to test whether the initial proposal would correspond the news article content to a type the same way as the dataset does.

Data

The dataset we chose fits our requirement in that it has the features we want, the size is big enough, it has diverse sources, and it is relatively new. However, the size is too big, to extent that it becomes inconvenient to process and use.

The dataset was retrieved from a GitHub release page. It is 10 files of zip multi-volume archive data, 9 of which have size of about 1 GB and the last is over 100 MB. After decompressing the files, the data is in a single 29 GB .csv file.

While processing data with Pandas, we encountered both memory and disk space errors. Even with the enormous size, processing the data with Pandas would need to read it into memory and save to another similarly sized file before we could delete the old file. This exceeds the resources a student account could have on GPU Farm Phase I and of course exceeds the limit on Google Drive and Colab. And since we only applied for GPU Farm Phase I, we spent a few days applying to Phase II for more resources.

We used Dask, instead of Pandas, to process the data. Owing to Dask's out-of-core characteristics, using its functions, such as `read_csv`, is more efficient and less likely to face a memory error, compared to their counterparts provided by Pandas.

The parameters and use of APIs provided by Dask are about the same as those provided by Pandas, yet the underneath implementation is far different.

We deleted columns `id`, `domain`, `scraped_at`, `inserted_at`, `updated_at`, `keywords`, `meta_keywords`, `meta_description`, `tags`, `summary`, `source` from the data because we believe these features do not have much to do with the nature or

authenticity of the news articles we are studying (e.g. `id`), they are included in other features (e.g. `domain` covered in `url`) or the data quality in that column is just too poor (e.g. `meta_keywords` is nearly empty for all articles).

After processing that data, we saved it into multiple files. Due to the sheer size of the data, we stored them into 459 files, each with size of about 60 MB. The total size of the processed data is about 25 GB, but it is more convenient to start with a much smaller file for experimentation.

We did not go further to do tokenization and numericalization, which are common steps in natural language processing, in this step, because they will be automatically handled by `TextBlock` of Fast.AI, the Deep Learning library we are using.

	type	url	content	title	authors
0	rumor	https://www.express.co.uk/news/science/738402/...	Life is an illusion, at least on a quantum lev...	Is life an ILLUSION? Researchers prove 'realit...	Sean Martin
1	hate	http://barenakedislam.com/category/donald-trum...	Unfortunately, he hasn't yet attacked her for ...	Donald Trump	Linda Rivera, Conrad Calvano, Az Gal, Lincoln ...
2	hate	http://barenakedislam.com/category/donald-trum...	The Los Angeles Police Department has been den...	Donald Trump	Linda Rivera, Conrad Calvano, Az Gal, Lincoln ...
3	hate	http://barenakedislam.com/2017/12/24/more-winn...	The White House has decided to quietly withdra...	MORE WINNING! Israeli intelligence source, DEB...	Cleavis Nowell, Cleavisnowell, Clarence J. Fei...
4	hate	http://barenakedislam.com/2017/12/25/oh-trump-...	"The time has come to cut off the tongues of t...	"Oh, Trump, you coward, you just wait, we will...	F.N. Lehner, Don Spilman, Clarence J. Feinour,...

Model

As said above, for the purpose of building a prototype, we only used one of the 459 files to test the initial idea. The file contains some 14,000 articles. There are limitations to this file's ability to represent the whole set of data, which we will talk about later.

For now, we would use this file. And we only studied the relationship between the `content` feature and its category at this stage.

First, we needed to train a language model.

```
dls_lm = DataBlock(blocks=TextBlock.from_df('content', is_lm=True),
                   get_x=ColReader('text'), splitter=RandomSplitter(0.1)
                  ).dataloaders(df, bs=32, seq_len=80)
```

We got the text data from the `content` column, the `TextBlock` automatically handles tokenization and numericalization and `RandomSplitter` splits the data into training and validation sets. Then, we put it in a `DataLoader`.

To convert the integer word indices from tokenization and numericalization into activations that we could use for our neural network, we would use embeddings. We fed these embeddings to a *recurrent neural network* (RNN) using an architecture called *AMD-LSTM*. The Embeddings would merge with embeddings of uncovered vocabulary. This was automatically handled by Fast.AI's `language_model_learner`.

```
learn = language_model_learner(
        dls_lm, AWD_LSTM, drop_mult=0.3,
        metrics=[accuracy, Perplexity()]).to_fp16()
```

The loss function used by default is cross-entropy loss, which is appropriate for our classification problem. The *perplexity* metric used here is often used in NLP for language models: it is the exponential of the loss.

Since the training time of each epoch is longer than some other cases, we tried to store intermediate model training results so that one error in the middle did not bork the whole training process.

```
learn.fit_one_cycle(1, 2e-2)
```

epoch	train_loss	valid_loss	accuracy	perplexity	time
0	3.915998	3.757433	0.347351	42.838306	08:58

```
learn.save('1epoch')
```

```
Path('models/1epoch.pth')
```

We could load the intermediate result later and continue training. `language_model_learner` by default calls `freeze` when using a pretrained model.

```
learn = learn.load('1epoch')
learn.unfreeze()
```

```
learn.fit_one_cycle(10, 2e-3)
```

epoch	train_loss	valid_loss	accuracy	perplexity	time
0	3.517755	3.473242	0.369883	32.241100	09:43
1	3.543658	3.539965	0.366176	34.465710	09:47
2	3.434362	3.529346	0.371195	34.101658	09:38
3	3.357048	3.493074	0.377160	32.886902	09:42
4	3.184052	3.447661	0.385416	31.426811	09:47
5	3.088798	3.406446	0.393076	30.157858	09:47
6	2.971667	3.378779	0.398499	29.334929	09:41
7	2.785853	3.356011	0.403602	28.674570	09:38
8	2.702044	3.349814	0.406020	28.497444	09:41
9	2.685668	3.355438	0.406083	28.658161	09:41

We saved all of our model except the final layer that converted activations to probabilities of picking each token in our vocabulary.

```
learn.save_encoder('finetuned')
```

Then, we could use that saved model to build a classifier.

```
dls_clas = DataBlock(blocks=(TextBlock.from_df('content'),CategoryBlock),
                     get_x=ColReader('text'), get_y=ColReader('type'),
                     splitter=RandomSplitter(0.1)
                    ).dataloaders(df, bs=32, seq_len=80)
```

Compared to the language model DataLoader, this includes the `type` column as `category`.

We then used Fast.AI's `text_classifier_learner` to create a text classification model. And loaded the finetuned language model.

```
learn = text_classifier_learner(dls_clas, AWD_LSTM, drop_mult=0.5,
                                metrics=accuracy).to_fp16()

learn = learn.load_encoder('finetuned')
```

We trained the model then.

```
learn.fit_one_cycle(1, 2e-2)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.628790	0.543213	0.846915	01:23

```
learn.freeze_to(-2)
learn.fit_one_cycle(1, slice(1e-2/(2.6**4),1e-2))
```

epoch	train_loss	valid_loss	accuracy	time
0	0.520186	0.405315	0.876618	01:39

```
learn.freeze_to(-3)
learn.fit_one_cycle(1, slice(5e-3/(2.6**4),5e-3))
```

epoch	train_loss	valid_loss	accuracy	time
0	0.390426	0.334031	0.897182	02:01

```
learn.unfreeze()
learn.fit_one_cycle(2, slice(1e-3/(2.6**4),1e-3))
```

epoch	train_loss	valid_loss	accuracy	time
0	0.315966	0.324922	0.897944	02:27
1	0.237817	0.318096	0.896420	02:26

And by now, we had a working news classifier.

Experiment

We wanted to know how the classifier performed. Remember we said that there are limitations to this file's ability to represent the whole set of data?

By looking at the number of articles of each type in the file we used to train our model, we found that the distributions of different types were off by a lot. Some types had thousands of articles while some only had a few dozens.

```
from collections import Counter
Counter(list(df['type']))
```

```
Counter({'rumor': 63,
         'hate': 256,
         'unreliable': 501,
         'conspiracy': 3453,
         'clickbait': 191,
         'satire': 200,
         'fake': 5184,
         'reliable': 202,
         'bias': 158,
         'political': 2231,
         'junksci': 586,
         '': 61,
         'unknown': 51})
```

The imbalanced data meant that the model probably worked better on test sets with similar class distributions. And indeed, it had this kind of results.

Since we only used 1 out of 459 for training, we tested the model on one file of data, which contained similar class distributions and it yielded some 80% accuracy. On other files that had very different class distributions, accuracy could drop to even about 7%.

The results told us that we could succeed from our initial idea of building a news classifier and that we needed to expand our training dataset for the next phase for better results.

We tested on building the model on a few more files, which combined had a bit more balanced class distribution, and the testing results improved then, with the lowest accuracy for any test dataset we chose at a much better 33%. We believed that it would continue to rise if we included more data.

We also noticed that the data was collected over months and the quality differed. So, using model trained on old data to test on new data would get worse results. In the next stage, we will mix the new and old data during training.

Limitations

There are limitations to our model. From the scope of our problem, we can only use the texts to decide the categorization of new articles. For some articles, humans could easily tell their nature and authenticity based on common sense and general knowledge. However, the classifier cannot think that way, so some of the easy-to-recognize evidence to a human is difficult to find for the classifier.

Also, the dataset has limitations. There is only one class for each news article. This is not in line with the reality. For example, the dataset has the `political` and `conspiracy` types. We know that a lot of conspiracies are related to politics, and yet the dataset only supports

single class, and the single class classifier we build could not learn from this. The model could learn more if we could have multi-class datasets.

Plans

In the next stage, obviously we will include more data to train a more powerful model. We will try to use as much data as we can. But the data size is huge. In case we could not use up all data in time, we will try to find and use more balanced, representative, and higher-quality files.

We will also need to include the additional information of titles, authors, and domains in our model. Since they are also texts, we will concatenate them before the content texts. The language model should pick them up without issue.

We will continue to test and improve our model. Also, we will try to make an application based on our model that could help ordinary people.

We value sustainability. We will provide documentations for the use and future development of the app and guidelines for future model update, in case there comes such a need when new and/or better dataset or better technology comes out.

Timeline

04/04 – 17/04: Modification and Retesting – Li, Huang
Given the test results, take turns to modify the model and conduct testing.

04/04 – 17/04: Building Web App – Huang
Build the web app and integrate the model in the backend.

18/04 – 19/04 – Final Testing – Li
Explore final bugs and limitations and record final testing results.

18/04 – 19/04 – Documentation – Huang
Write documents for the application and provide guidelines for future development.

20/04 – 21/04: Making Video – Li
Record and edit the video for presentation.

22/04 – 24/04: Writing Report – Li, Huang
Write the report for submission.

24/04: Submission of Final Work