Fake Review Classification

Based on *Creating and detecting fake reviews of online products* by Salminen et al, 2022

Group Members: Ricky Zhong, Alice Marbach, Peter Van Katwyk

PURPOSE

Develop a data processing pipeline and model that can determine whether a review left on a product on Amazon is real (written by a human who bought the product) or fake (computer-generated).

INTRODUCTION

Consumers looking for products to purchase online often decide which product to buy based on the number and quality of reviews that are left. For this reason, seller accounts on popular websites such as Amazon selling products with the most reviews within the category have extremely high earning potential. Because of this, services have been created to synthetically bolster the number of reviews left on a product, usually by purchasing a fixed number of reviews for a price. Fake review detection has been an important task for companies as the purchase of fake reviews undermines the value of a review-based system.

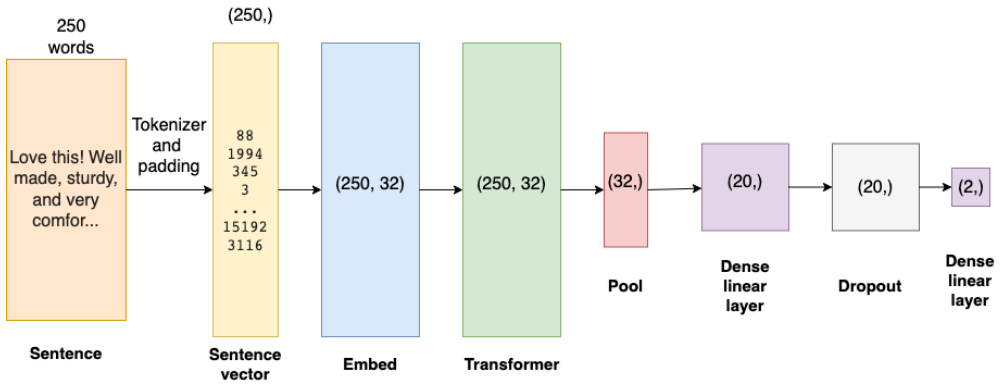
Our study implements a similar framework as that presented in the paper “Creating and detecting fake reviews of online products” by Salminen et al (2022). In the paper, the authors create a dataset with real reviews taken from Amazon’s public repository as well as fake reviews generated by GPT-2. They then fine-tune the RoBERTa model to accurately identify which of the reviews are real versus fake. Our implementation includes the transfer of code-base from PyTorch to Tensorflow of the same fine-tuning process, as well as the comparison of our own simpler transformer model, FRDetector, with the fine-tuned equivalent presented in the paper.

METHODS

Data: The dataset contains 20,000 real reviews and 20,000 fake reviews and takes up 15 MB in total. The real reviews come from real reviews left on Amazon products (Amazon Review Dataset). Fake reviews were generated using DeepMind’s GPT-2 model that was fine-tuned to the Amazon Review Dataset. This way the synthetic samples generated are seemingly close to real reviews.

Preprocessing: In fine-tuning the RoBERTa model, the Transformers library was used to tokenize and pad the data. The embedding matrix was downloaded and words were converted to integer ids to be inputted into the model. In the case of our own transformer model, the same tokenizing and padding took place using TF Keras preprocessing tools. The labels were then encoded from “CG” (computer-generated) to 0 and “OR” (original) to 1.

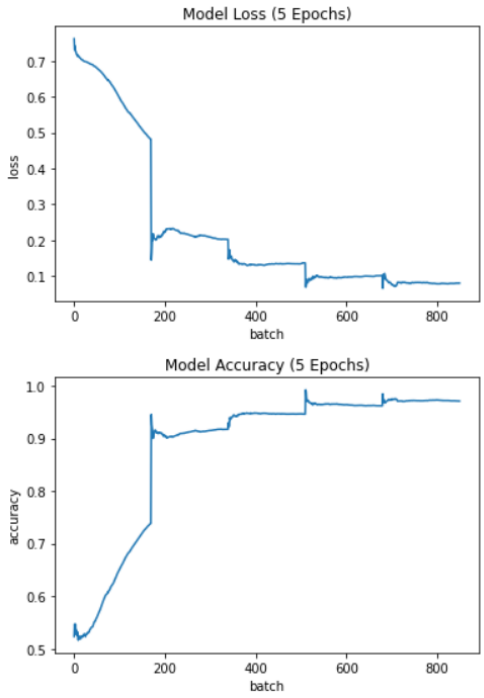
Model Architecture: We compared the performance of two different architectures: the pretrained TFRoberta Transformer model, and our custom architecture FRDetector, which was a more simplified transformer architecture and used multi-headed attention. See the figure below for the architecture of the FRDetector model:



RESULTS

Our results were strong and conclusive. Using our custom-built RoBERTa Model, we were able to achieve a surprising peak validation accuracy of 0.9227, or 92.27%, with just 5 epochs. As shown below:

```
Epoch 1/5
170/170 [==============================] - 20s 65ms/step - loss: 0.4819 - accuracy: 0.7384 - val_loss: 0.2380 - val_accuracy: 0.8972
Epoch 2/5
170/170 [==============================] - 9s 51ms/step - loss: 0.2029 - accuracy: 0.9173 - val_loss: 0.1932 - val_accuracy: 0.9227
Epoch 3/5
170/170 [==============================] - 7s 44ms/step - loss: 0.1370 - accuracy: 0.9462 - val_loss: 0.1950 - val_accuracy: 0.9203
Epoch 4/5
170/170 [==============================] - 7s 44ms/step - loss: 0.1018 - accuracy: 0.9620 - val_loss: 0.2130 - val_accuracy: 0.9219
Epoch 5/5
170/170 [==============================] - 7s 44ms/step - loss: 0.0808 - accuracy: 0.9709 - val_loss: 0.2369 - val_accuracy: 0.9186
```



Based on the original paper we based our model on, humans can determine fake versus real reviews in this dataset with anywhere between 52 to 65% accuracy, and a standard ML model reaches about 86% accuracy. The fakeRoBERTa model implemented and used by the original paper, which takes 2:34:16 to run (compared to the 0:54:12 for ours) achieves an accuracy of 0.9664, or 96.64%.

Overall, our model very much exceeded our expectations and performed similarly to the original pretrained RoBERTa model despite being scaled down quite significantly.

CHALLENGES & DISCUSSION

We ran into a few issues throughout the time we spent building and fine-tuning our model. The primary issue was in translating the PyTorch code based on the original paper to our own simpler Tensorflow implementation. We quickly learned that several functions and libraries do not have direct tensorflow equivalents, and we had to do a lot in terms of building our own preprocessing structure as adapting the tokenizers to our custom architecture.

Our model certainly has its limitations. Beyond not being as accurate as the original model, there are concerns with the data itself as well. The data used in our training for the Amazon

reviews are all in English and thus would not support an international audience or community, and the computer-generated fake reviews used to train the model are ultimately based off of mimicry and conglomerated information rather than creativity, meaning that it's entirely possible the model starts failing quickly with even small shifts in review "culture" or any kind of creative change.

Although our model is extremely accurate, it is worth considering that the extra ~5% accuracy of the original model from the paper could be worth the extra complexity and time needed to run it and produce predictions.

REFLECTION

- How do you feel your project ultimately turned out? How did you do relative to your base/target/stretch goals?

We felt that the project turned out better than we expected. There were unforeseen difficulties in transfer from PyTorch to Tensorflow, particularly in the implementation of the Transformer Tensorflow functions as opposed to PyTorch, but we were able to achieve all of our base, target, and stretch goals.

- Did your model work out the way you expected it to?

At first the model did not work the way we expected. An issue that we ran into that we had not anticipated or seen before was the OOM memory errors that we received when we made our architecture too large. But, through some simplification and tricks in preprocessing, we were able to capture the classification with a simpler model. In the end case, the model did work like we expected.

- How did your approach change over time? What kind of pivots did you make, if any? Would you have done differently if you could do your project over again?

Throughout the project our approach shifted due to different challenges that arose. For example, we all expected to take the majority of time carefully crafting our own architecture, whereas in the end we spent a lot of time formatting the data and transfering code from

PyTorch to Tensorflow. We did pivot in that we initially thought we were only goin to recreate the paper, but instead we decided that we would create it and also try to make a model on our own. If we could start over again we would avoid any code written by others and start on our own. I think we could have avoided some issues if we were writing our own code rather than trying to get code from the paper to run that someone else wrote.

- What do you think you can further improve on if you had more time?

I think our architecture and training procedure could be greatly improved. Because so much time was spent in preprocessing and transferring from PyTorch to Tensorflow, we didn't spend as much time trying all different types of architectures and hyperparameters. With more time, we could have fine-tuned our own model a little bit more.

- What are your biggest takeaways from this project/what did you learn?

One of the biggest takeaways from this project was that a lot of effort goes into making a machine learning project work. In this class we talked a lot about architectures, which really is the fun part, but so much needs to happen with the data, the formatting, the code, and more to even be able to apply the algorithms.