Final Project Report

Who Said It?

Edgar Allan Poe vs Shakespeare

Group 3

Introduction

Our project is on Question Answering in the context of Natural Language Processing. Question answering is a field of natural language processing that generates answers to user questions by searching for the answer to that question in some relevant body of information.

Problem Definition

Our problem is that we want to recognize differences in writing styles by choosing to focus on the writings of two authors, Edgar Allen Poe, and Shakespeare.

Project Goals

Original Promises/Goals

- Research use of GPT-2 to implement for this project (Complete by April 27)
- Import and set up dataset for processing (Figure out how to create individual sentences from the books) (Complete by April 30)
- Set up training parameters (Complete by May 3)
- Train the model on the data sets (Complete by May 8)
- Feed questions to test models and evaluate results (Complete by May 13)
- Design and user interface (Complete by May 18)

Goals Met

- Researched GPT-2 for question answering. Upon completion of research made the decision to switch to Huggingface Transformers trained on Distilbert pre-trained model for sequence classification.
- Used PyTorch to set up training parameters for our model

- Trained the model on our data sets and optimized the model during fine-tuning
- Tested the model on test data and verify results
- Input sequences to the model after testing and analyze results
- Create a human coded UI chat bot which will receive as input any sentence of our choosing and quickly return an answer to us indicating which author the model believes is more likely to have written that sentence.

Contributions:

Matthew ~ First, I researched the GPT2 model, running a tutorial we found on classifying movie reviews to see if it would be a viable choice for our project. Then, I helped with adding poems to our dataset in Google Drive, a task accompanied with a small amount of text processing (replacing �s found in the poems with quotation marks where necessary). Finally, my largest contribution to the project was optimizing and fine tuning our model based on results given from testing multiple inputs. This is what I signed up for in the project proposal: working with input and result evaluation. I was able to look at how the model performed on our test dataset, and fine tune our training arguments accordingly. Originally, the model was not very accurate with the test dataset but as I made changes to reduce overfitting in the model, we were able to greatly improve our accuracy with both authors.

Kyler ~ The first steps I took in the project was researching GPT2, and how to build a question answering model. Since we wanted to categorize a poem into either styles of the writers, I started doing work with a GPT2 Finetune Classification. After our initial set of data I started doing work for training the Model, which was my original assigned task of “Will mentor with training the model on the data sets.” from the project proposal as well as the bulk of my work. This involved research on huggingFace Transformers, which led us to using pyTorch and DistilBERT. Some more work I did was researching the use of gpu and saving models to save us hours during training. Soon after this we realized there may be issues with our datasets, so I sourced approximately 20 more poems from Edgar Allen Poe which doubled his number of lines. This led us to having more accurate testing results.

Katie ~ I began by researching the use of Open AI’s GPT-2 Question Answering Model. It became clear that the way we defined our problem was more aligned with a classification problem than an explicit question answering problem. Therefore, I began searching for resources that would help us combine GPT-2 Question Answering with classification. Ultimately, I came across the Huggingface Transformers documentation at the recommendation of the instructor. To get the project going I dealt with handling the input data from Luke and preparing it for our training model. After that, I incorporated code referenced from Huggingface’s Fine-Tuning Custom Data Sets for Sequence Classification documentation. That resource recommended the use of PyTorch for training the model and setting up the training arguments. I helped set up the initial model and training parameters that would be used later for training and testing. Later on, I helped analyze initial testing on all sequences of our data set, coming to the conclusion that the data was skewed. Later on after testing I helped write a small piece of logic to connect the testing functions to be called from the UI as needed.

Luke ~ I started my work on the project by researching GPT2, particularly using it for classification; when we decided to switch to pure Hugging Face I started reading their documentation for research. I helped with gathering poems from various sources to build our corpus. The biggest contribution I made was with preparing the corpus for the model. Me and Katie worked together to import the documents, I then did additional cleanup on the lines of the poems and split up individual lines and gave labels to the Poe and Shakespeare lines; finally, me and Katie split the list into train, validation and testing datasets and encoded them.

Ryan ~ I first started by learning how to use GPT-2 before we eventually switched to using Hugging Faces. I then started writing the code for the user interface for our model. Originally it contained a menu for selecting whether you wanted to enter a sentence or quit, with the infrastructure available to add more menu options.

After making this UI, I then moved on to working with some of the group members to figure out how we could send a single sequence through the neural network, and get the results. During this process I found that we had issues with our current method of splitting our data set into sentences for Poe's data set. I thought of how we might fix it, worked on implementing it some on my own, and then worked with Luke, and then passed off the task to Luke. After more/different data was grabbed for each of the authors, I went in and did some additional cleanup, removing things such as dedications and quotes at the beginning of works.

After much work and research, but no luck in finding how to run a test sentence through the model, Katie re-sent over a tutorial for using hugging faces, which I had read before, but hadn't fully understood. I re-read through it and finally figured it out, and then applied a modified version of the tutorial's process for testing in our code. With this I created the base for testing our model, giving us the ability to run a single item through the NN. Other group members repurposed this base for use in 'for loop' based testing, and I used the base, single item version, for the connection to my UI.

Later when using the UI to test the model and prepare myself for demonstrating it, I found interfacing with the menu to be obtuse for our purposes, so I removed the menu and reworked the UI. I made it so the user could continually enter sentences and then quit by entering a blank line. The new UI also utilized a line code that Katie had written for me, indicated in the code. I also wrote part of the conditional code which determines whether to run the code for training, or to load the model for testing.

I mentored the design by figuring out the type of NN we would need to complete the job and advising my teammates and helping them when they had questions on the design of the model. I also worked on various parts of the model, as described above. I was also often delegating tasks to my teammates regarding what needs to be worked on the model next, whether that be an issue I found or just the next task that needed to be completed. I mentored on the UI by creating it.

Solution/Model Formulation:

The first step we took after importing our data was to split their texts into a one-dimensional list for each author. These sentences were split on end-lines and ending punctuation like question-marks and exclamation-points. Next we split and tokenized the data for use in training and testing the model. We then moved to training the model. Originally, we were using the default values for weight decay (0.01) and learning rate (5E-5). With those parameters in place, we then worked on finding a number of epochs that minimized the training loss without costing too much RAM and disk space. We came across 15 epochs as a good balance. However, the model was not super accurate on our test dataset at this point so we decided it needed some fine tuning. Upon revealing other metrics such as validation loss and accuracy, we discovered that our model was overfitting to our dataset, so we set to work on generalizing. This included multiple hours of testing at which point we landed on a weight decay of 1.0, a learning rate of 0.000001 and 10 epochs as the optimal training parameters. This provided us with loss values that were relatively low and approached the same value. We also had an above 80% accuracy on our validation dataset, and much improved accuracies on our test dataset.

Implementation:

- Beginning: In the first few lines of the code we download dependencies and import some of the needed libraries. We also mount the Colab notebook.
- Text Processing: Here we import our files of text Poe and Shakespeare and split up the poems by line into two lists, one for both writers. Next we split up Poe's lines on '.', '?', and '!' to handle some of Poe's longer sentences; we decided to only split Shakespeare's lines farther on '.' because these '?', '!' made less of a difference in his writings. In this stage we also removed empty lines from both lists and provided labels, 0 for Poe and 1 for Shakespeare, based on their new line counts.
- Split Data: This is the stage in which we split the data first into a train test split, making sure to keep the appropriate labels with the correct lines. We then complained the test sets for Shakespeare and Poe, with their labels, and split this combined dataset into training and validation datasets
- Tokenize Data: Using bertTokenizer we created line encodings of our three datasets, one for training, one for validation and one for testing. We store these encodings with their associated labels using the poeShakespeareDataset class.
- Training the Model: Next, we define our model and each of the arguments that will be used during training. These were optimized during fine tuning in order to provide the best results. If we already have a trained model when running this code, it will simply load the saved model.
- Testing the Model: We then start to feed the model lines from our test dataset. Our model's output is a percentage confidence for each author that the line was written by one or the other. We then average the

percentages from all lines by Poe and all lines by Shakespeare, so that we get two values that we can easily compare.

Computational Analysis:

Originally, we were training using high numbers of epochs and a low number of logging steps. This meant that the model was training for a long time, and at very short time intervals the model would display metrics on the current step and also output those data to a directory. This involved using a lot of RAM and disk space. In order to improve on that front, we increased the logging steps so that the display would not be updated as often and the disk would not fill up with logging metrics as quickly. We also switched to using Google Colab's GPU as it improved runtime by a considerable amount (from 90 min to approx. 15 min).

Conclusions:

Our model was able to accurately detect poems that were written by Shakespeare (on average the model was 92% confident they were Shakespeare's), however it was much less accurate when Edgar Allen Poe was used as input (on average 54%). In conclusion we discovered that data is one of the most important aspects when it comes to training models. If we could have better processed our data and found many more sources then we could potentially have had more accurate testing. This could have also been approached by using more contrasting styles, perhaps like Dr. Seuss or other more contemporary authors. Testing with different Models for classifying and question answering would be one possibility for us to look into.

References:

- https://lucidworks.com/post/what-are-question-answering-systems/ (History)
- http://staffwww.dcs.shef.ac.uk/people/M.Greenwood/nlp/pubs/thesis.pdf (History)
- https://huggingface.co/transformers/custom_datasets.html (Trainer and Torch Object)
- http://www.public-domain-poetry.com/stories/edgar-allan-poe (Edgar Allan Poe Poems)
- https://www.poetryloverspage.com/poets/poe/poe_ind.html (Extra Edgar

Allan Poe Poem Source)

- http://www.public-domain-poetry.com/william-shakespeare (William Shakespeare Poems
- https://www.thepythoncode.com/article/finetuning-bert-using-huggingface-transformers-python (Display Metrics)
- https://towardsdatascience.com/optimizing-neural-networks-where-to-start-5a2ed38c8345 (Optimization)
- https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/ (Good Fit Graph)
- https://huggingface.co/transformers/usage.html (Tutorial used for learning how to run data through the trained model and get results)