Machine Learning Based Chat Analysis

Christopher Brousseau, Justin Johnson, Curtis Thacker

Abstract

The BYU library implemented a machine learning based tool to perform various text analysis tasks on transcripts of chat-based interactions between patrons and librarians. These text analysis tasks included estimating patron satisfaction, and classifying queries into various categories such as Research/Reference, Directional, Tech/Troubleshooting, Policy/Procedure and others. An accuracy of 78% or better was achieved for each category. This paper details the implementation details and explores potential applications for the text analysis tool.

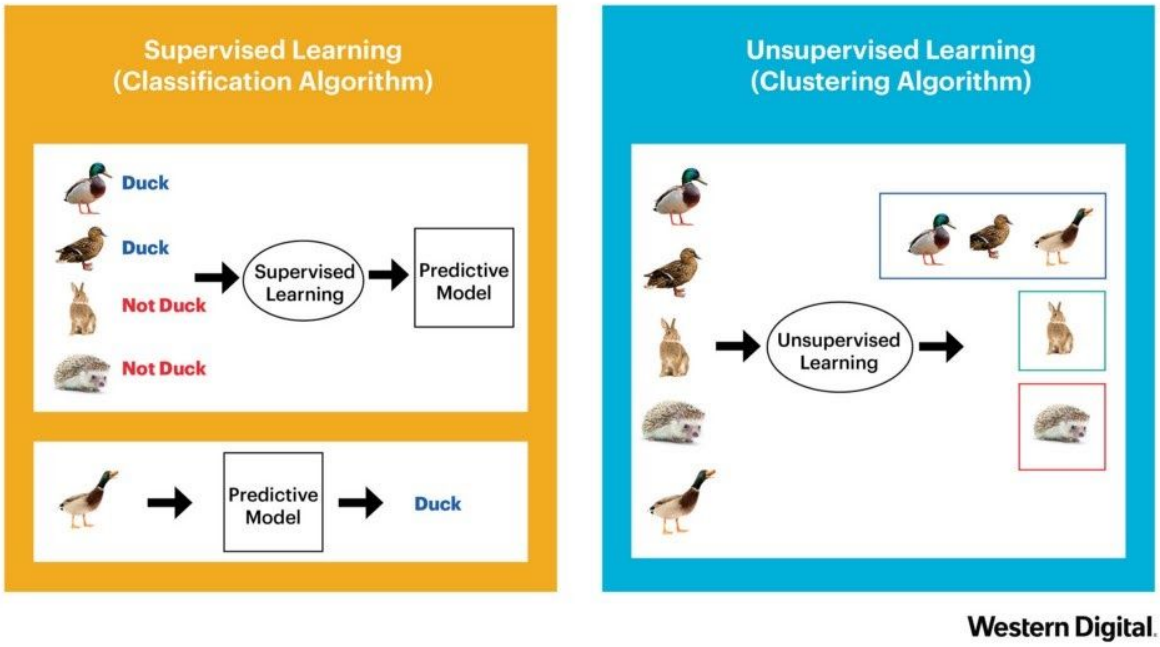
Introduction

In 2019 Whitchurch and Merrill published a qualitative analysis of chat transcripts between librarians and patrons of Brigham Young University's Harold B. Lee Library (Whitchurch and Merrill 2019). Their dataset was created by coding 4,475 chat transcripts recorded during 2016. They coded into 25 categories - described below. We attempt to create a machine learning model that will programmatically code a chat transcript. Using the resulting model we hope to programmatically code about 9 years or 70K additional transcripts. This larger dataset would allow for analysis of trends over time.

Background

Background on Machine Learning

"Machine Learning is the study of computer algorithms that improve automatically through experience" (Mitchell 1997). A machine learning model is an algorithm that used patterns in data and previous examples to learn to perform a task. Detecting a cat in an image and predicting how much a 3,000 square-foot house in Boston will be worth in 2025 are examples of tasks that can be performed by using machine learning.



Supervised Learning (Classification/Regression) | Unsupervised Learning (Clustering) | Credits: Western Digital [13]

There are two basic paradigms for machine learning models (hereafter called models), supervised and unsupervised learning. A supervised learning model uses patterns found in example data to find relationships between inputs and outputs such as whether a cat is a given picture. Unsupervised learning algorithms are used in pattern detection and descriptive modeling tasks. These algorithms do not have preexisting

Background on Text Classification

For our experiment, there are two tasks to accomplish. The first task is very similar to Sentiment Analysis. Sentiment Analysis[1] is a classic Natural Language Processing (NLP) task which tries to predict the overall positivity or negativity of a statement or utterance. Our experiment required us to use text to provide a variety of labels for a block of natural language text.

NLP is a broad field in machine learning, spanning from text processing to speech recognition. One of the main focuses and most common tasks of NLP is creating what are called language models. Language models are essentially large records of computer-generated rules pertaining to a language, and there are many different types of language models. In order to expedite our

[1] For a demo that showcases several different ways to do sentiment analysis along with an easy-to-understand visualization, please visit https://quicksentiment.herokuapp.com.

efficacy, we've opted to use a pre-trained language model for our task, and fine-tune or adapt it to our specific dataset. BERT is the base model that we've chosen for this task.

Other very common tasks in NLP are text processing, part-of-speech tagging, tokenization, and sentiment analysis, some of which we are using as part of our model. Text processing or pre-processing involves cleaning data so that it can be used without introducing any unnecessary bias, for example in our dataset, all of the customer interactions originally contain ip addresses and other incriminating data that isn't relevant to the task we're trying to complete, and would introduce bias into the results that we aren't prepared to deal with, like the idea that only a certain ip address ever asks questions about certain topics. Tokenization is the act of splitting up a text by tokens, in our case individual words. In other cases, those tokens could be morphemes or prefixes/infixes/suffixes, etc. Some language models are based on tokenization, such as the Bag of Words statistical model, which counts each unique token in order to draw important conclusions from the text. Others only utilize it like the Term Frequency-Inverse Document Frequency model, but it's safe to assume that where a language model exists, some form of tokenization was done. We did not use part-of-speech tagging in our model but it is often used to introduce bias for grammatical structure into language models, usually for sequence-to-sequence models used for translation or summarization.

Back to the first task, sentiment analysis combines several of the NLP methods above in order to complete its task. It's a corpus-based method, meaning that it's either needed to make our own corpus of "known" words or to piggy-back off of an already created corpus. In our case, we use both methods. We utilized BERT's pre-trained word embeddings as a corpus, and during fine-tuning whenever we came across a new word that wasn't in the current corpus, we added a new vector to represent that word. These vectors or word embeddings are based on how frequently the word shows up, but also take into consideration the word's placement in a sentence, along with its perceived semantic meaning. All of these are taken into account for determining the overall positivity/negativity of a given word. Once we have those numbers, to analyze sentiment, we add up each individual word's "score," and from there determine how positive/negative the whole utterance is. We aren't analyzing straight sentiment, although our model has learned to consider the overall sentiment of a chat as a statistically significant factor in determining the satisfaction of the patron.

Our second task is Classification which is predicting which of two or more categories (or classes) a given input fits into. For the related task of Binary Classification prediction is limited to just 2 classes such as true or false, cat or dog, black or white, dead or alive, etc. We've opted for a black-box model, as opposed to other more easily understandable models for classification. Instead of manually injecting bias by predetermining rules for the classification to follow, we allowed the language model BERT to follow its algorithm to "learn" how it should classify each input.

BERT (Jacob Devlin et al 2019 Bidirectional Encoder Representations from Transformers) is a sophisticated unsupervised machine-learned model created by Google. Its purpose is to take natural language in plain text and represent it as a sequence of numbers that capture the nuance and meaning of the language. Doing this enables the creation of other models that perform tasks like sentiment analysis and translation. For all the tasks described in this paper, we used DilBERT (Victor Sanh et al 2020 A distilled version of BERT), a smaller version of BERT that doesn't sacrifice accuracy or efficiency.

Background on Library Chat

Chat or virtual reference is an effective way for libraries to provide service and assist their patrons. With this importance, libraries are looking for ways to improve the service they are providing. A review of the literature shows that libraries primarily do this in two ways, first, by making technological improvements to the service and second, by evaluating the service and improving it through training and support to library employees.

Some examples of libraries making technology changes to improve the service include using proactive chat widgets or chatbots. A chat widget is a window that opens the chat system within a webpage, kind of like a pop-up. It prompts the user to see if they need assistance. Many libraries are starting to implement this type of function in a proactive way so that patrons are aware that an online chat exists. Bowling Green State University made a move in this direction with proactive (pop-up) chat widgets embedded within their library webpages, catalog, and databases. Since implementation, they have seen the number of chat reference questions received more than double (Rich et al. 2018). The University of Texas at Arlington also implemented a proactive chat widget and saw their answered chat questions double from 4,020 in 2015-16 to 8,120 in 2016-17 (Pyburn 2018).

A chatbot is an interactive computer application that responds to sentences in a meaningful way (Allison 2011). It uses AI to respond to questions like a human and identify the need and escalate the patron to the appropriate source or location to receive answers. They can be used to handle routine or repetitive tasks to common questions, such as 'when does the library close', or 'how do I find an item'. The University of Oklahoma library uses a chatbot called "Bizzy". It assists students in finding answers to common questions and helps in finding resources. The chatbot can also assist by providing responses to questions asked using natural language rather than requiring a structured (author/title/subject) search such as in the library search bar (The University of Oklahoma Library …[updated 2020]).

The second way to improve that was mentioned above is doing a qualitative analysis or evaluation of the chat service. This is generally done via a review of chat transcripts using an established rubric or dataset to evaluate all chats in the same manner. There are many publications that demonstrate the usefulness of this approach and describe the method of coding typically used for these evaluations. The studies range from reviews of the level of staffing and its effects on service, the use of RUSA or ACRL frameworks for reference

interviews, the types of questions being asked, the type of language used in the interactions, and identifying training needs for staff. Of these examples, the following are included in more detail to show the methods used and the similarities between the studies. These evaluations are one review of the interactions based on established reference frameworks such as the ACRL framework (Hunter et al. 2019), and one study identifying answers through the evaluation of the questions being asked ().

In 2009, the Library at the University of Guelph did an assessment of their virtual reference and instant messaging services. To do this they reviewed all transcripts from the past two years and categorized them into five broad categories of the type of questions asked. These categories were directional, policy, ready reference, specific search, and research questions. These categories were further broken down into 39 subcategories. The process involved a team reviewing the transcripts and coding the interactions according to which categories they fit within. Upon finishing this, they analyzed the results to determine the types of questions that were commonly being asked so that the library could make an informed decision about whether to keep the two services. Knowing how the services were being used and the types of questions asked in each service would be helpful in making this decision. Their results found that each service was assisting different types of patrons seeking assistance in different ways.

In a study published in 2019, a task force from Berkeley College reviewed 369 chats from the fall semester of 2017. They focused on chat interactions that were identified as needing research or writing help. With this focus, they analyzed each of the interactions according to the ACRL's Framework for Information Literacy in Higher Education. They did this by establishing a method rubric to direct the task force in its analysis and coding of the chats. In analyzing the results of the study, they started looking for best practices in training librarians to conduct virtual reference within the ACRL Framework.

These examples included above show how libraries have typically conducted qualitative analysis of chat interactions through the use of teams or task forces using established rubrics and conducting coding of each transcript. These demonstrate the potential benefits that are available in creating a machine learning or sentiment analysis model to assist in this type of analysis. There has been some use of sentiment analysis within libraries already such as a study that evaluated LibQUAL+ surveys open ended comments (Moore 2017) and one that did a sentiment analysis of quality surveys for online classes and library use. In the LibQUAL+ study, the author created a sentiment model using manually coded data for its dataset to identify positive and negative comments within the survey. He took 514 coded entries from five separate surveys to create a set of positive and negative word vectors for the open comment sections of the LibQUAL survey. He then ran the comments from those five surveys through his model to review the results. Expansion of work like this can be applied to other word-based interactions or transcripts such as chat.

The second study….

These are examples of how libraries can use an established dataset to create a qualitative analysis of chats through sentiment analysis.

Project Description

We performed supervised learning on 21 different tasks, one of which was fine-grained sentiment classification, the other 20 being binary classifications. We've achieved really quite impressive results on all of these tasks, and we've been able to synthesize some analyses similar to Michael Whitchurch's project before ours, but without needing human researchers. We've been able to analyze and classify 77,000 chats in 21 hours; effectively 100 times faster than Michael Whitchurch's research project went. We expect this vast increase in speed to be valuable to anyone who would like to implement this project for themselves.

Our experiment makes use of three python code libraries: Pytorch, Transformers, and FastAI. We used the transformers library to access DilBERT from HuggingFace🤗. Using Transformers version 2.5.0, we have easy access to the three portions of the model we need most, the pre-trained model, tokenizer, and config. The DilBERT model has a maximum input sequence length of 512, which isn't immediately compatible with FastAI's architecture, which is built more for working with an RNN than a transformer model. Beyond that, it's much shorter than many of the chats we were attempting to analyze. We had to implement a custom wrapper for the tokenizer to normalize the input and the max sequence length. This allowed us to make our model as flexible as possible.

In order to get our model to correctly process our data, we needed to first preprocess the text, then create a way to load it into the model. To properly load our data into the model, we rely on the FastAI DataBunch API, which gives us an easy way to not only load the data but also shuffle and test it without requiring too much configuration. You'll remember from the brief explanation of APIs that they define standards for input and output for different types of data, and one of the reasons why this DataBunch API is helpful for us, is that it accepts so many different types of data, and presents the same types of output, meaning that our model is very flexible as to what types of data and what formats it can work with.

Based on this excerpt (HuggingFace🤗 2020), the creators of the Transformers library, all of their Pytorch-based transformer models output tuples with elements that differ across models. Because we're hoping for flexibility between models, we don't really care about receiving all of these elements, and adjusting our whole architecture for each and every possible model would not be useful for our purposes, so we created another custom wrapper that allows us to access only the logits no matter which transformer model we choose to put through it.

In the last part of the setup, you'll remember that we are working with the 21 data categories defined in Michael Whitchurch's experiment. Some of these are binary classification, while others are multiclass, meaning that for testing, we need to be able to seamlessly transition

between the two tasks. Thanks to Transformers, we have easy access to these features in the form of the config for each model.

The model is now set up correctly. Because of all of the work we put into custom wrappers, our program works not only with DilBERT, but also BERT, RoBERTA, and XLNet (all large pre-trained language models). We are training on the data collected and labeled by Michael Whitchurch, and predicting all of the same categories. We are using AdamW as an optimizer, and again FastAI allows us to bundle our model up nicely with the Learner API.

FastAI allows us to visualize our data immediately and easily after every epoch. To analyze the results, ClassificationInterpretation is a useful API to implement, allowing you to generate confusion matrices quickly and effectively. After that, we are able to predict any of the classes we trained on in our model, along with saving our model to use within larger systems and output our predictions as files. Saving the model is especially important, as it significantly reduces prediction time, as we don't need to retrain anything.

Dataset

Our dataset consists of 4476 chats collected in 2016. This data was collected through the Harold B. Lee library's chat system and analyzed by Michael Whitchurch's team and detailed in his publication. It contains 21 categories of questions that Whitchurch found to be useful in determining a couple of things: are patrons generally being satisfied by the chat experience, and how do trained library professionals compare to students when interacting meaningfully through the chat. We extended the first of these questions to our project, utilizing the same categories present in the original. The data originally came in an unusable format for our model. In order to clean it up, we first replaced all personal information that came with the chats to protect our users' privacy. Now, instead of displaying StudentID@IPAddress: and LibraryID@IPAddress:, it displays PATRON and LIBRARY. The next step in cleaning was replacing all of the text answers for the categories with numbers, so instead of TRUE or FALSE, it was 1 or 0. Once our data was cleaned, we were ready to begin training and testing. Something worth noting is that despite all of the effort put into cleaning the dataset, there were still several biases that inhibited the model from

Here's a breakdown of the data on question types and how they are portioned throughout the entire dataset:

Question Type	**Total**	**Percentage**
Research/Reference	2936	65.6%

Policy/Procedure	124	20.4%
Tech/Troubles	397	8.9%
Directional	314	2.8%

Lastly, here's a breakdown of each of the 21 categories tested in detail:

	Label	**Scale**	**Meaning**
1	Patron Satisfaction	1-5	Overall satisfaction of the patron
2	Unanswered	True/False	Whether or not a librarian or student answered the chat
3	Check Original Chat	True/False	Whether a given chat was the original contact, or if another chat needed to be referenced
4	Premature exit	True/False	Whether the patron left before the question was answered
5	Cite source(s)	True/False	Whether a patron was citing a source from the library
6	Guided to source	True/False	Whether the patron was successfully guided to the source
7	No source	True/False	Whether the source actually existed at the library
8	Unnecessary	True/False	Whether the source citation was actually necessary
9	Research/Reference	True/False	Question type - Research question
10	Directional	True/False	Question type - Directions to something on campus/in the library
11	Tech/Troubleshooting	True/False	Question type - Help with using or debugging technology
12	Policy/Procedure	True/False	Question type - Help understanding what the rules are and why they're in place
13	Inappropriate	True/False	Whether the patron behaved inappropriately or used the chat for an unintended purpose
14	Student-to-student	True/False	Whether the patron (a student) was connected

			to a student employee through the chat
15	Greeting	True/False	Whether a greeting was expressed at the beginning of the chat
16	Follow-up	True/False	Whether any follow-up is required based on the chat
17	Closing	True/False	Whether a closing statement (e.g. Bye) was expressed at the end of the chat.
18	Campus question	True/False	Whether a question is about a campus schedule or activity
19	Perceived inaccurate	True/False	Whether the answer given by the librarian is completely accurate
20	Perceived incomplete	True/False	Whether the question needed more information than answer given by the librarian contained
21	Employee Inappropriate	True/False	Whether at any time the employee behaved inappropriately during the chat, or if they used the chat outside of its intended purpose

Results

Our project was largely a success, and we expect that anyone who implements this model will experience the same significant margin of improvement over more traditional methods of analysis. This project was very useful for our purpose, which in this case was providing a means for quick and accurate analysis of customer interactions. This analysis provides a bunch of meaningful data for a variety of purposes including employee training and HR management, customer satisfaction and quality assurance, and overall efficiency. This data can be used as the basis for a custom chatbot specific to a company's data and market, or as a helpful start to building better-customized training for employees. It's main usage, however, is beginning a company's journey towards taking control of their customer interactions.

Our current dataset has several problems with it, however, none of these are hard-to-solve. In building a custom dataset for fine-tuning on your own data, at least a couple thousand examples of labeled data are needed, be those chats, reviews, or just general email interactions. Our model accepts CSV, XLSX, or TSV formats. You should only have to run through training one time to get accuracy in the high 80's or low 90's with a good dataset.

Satisfaction confusion matrix X - Predicted Y - Actual

	Dissatisfied/Frustrated	Neither	Satisfied	Above and Beyond
Dissatisfied	~2%	<1%	<1%	0%
Neither	0%	8%	4%	0%
Satisfied	0%	~2%	82%	0%
A&B	0%	0%	<1%	~2%

Employee Appropriateness confusion matrix X - Predicted Y - Actual

	Appropriate	Inappropriate
Appropriate	100%	0%
Inappropriate	0%	0%

Training data for Satisfaction

Epoch	Training Loss	Validation Loss	Accuracy	Error Rate	Time Elapsed
0	0.357155	0.375009	0.891374	0.108626	00:39
1	0.320672	0.391226	0.891374	0.108626	00:41
2	0.232104	0.412163	0.900958	0.099042	00:37

Training data for Employee Appropriateness

Epoch	Training Loss	Validation Loss	Accuracy	Error Rate	Time Elapsed
0	0.000000	0.000000	1.0	0.000000	00:25
1	0.000000	0.000000	1.0	0.000000	00:29
2	0.000000	0.000000	1.0	0.000000	00:23

With the results, we can look at trends over the last few years based on the criteria we used. We can identify potential issues based on the results and use these for the purpose of evaluation or improved training. Being able to fine tune the results can also help in identifying some of the underlying issues that might be present in the service being provided. Fixing these issues is an opportunity to improve a service by understanding better how the interactions have progressed. An example of this is using the satisfaction rating to see if there have been any trends in our ratings in chat interactions and identify potential issues that might be present within the service being provided. Looking at graph 1, you can see that our number of satisfied chats have been decreasing the last three years. While our total number of chats have decreased as well, this alone does not account for the decrease. Comparing ‘Neither’ and ‘Dissatisfied’, we can see that these two categories have stayed roughly the same across the three years. In comparison, the ‘Satisfied’ category is seeing a marked decrease. Identifying this trend can help in finding areas for improvement in our chat services. This can be done through improvements to the service and to training of the employees who handle all the chat interactions. With the results available we can spot the trends and do more in-depth analysis to improve our service.

Two other categories we can look at for potential value are the unanswered group and the wait time. Using these results, we can track how long it is taking for chats to be answered and identify the number that go unanswered. These are areas we can conduct focused improvement through training by working to answer chats and make sure that they do not go unanswered more quickly. For example, in 2018 we had 1422 chats have a wait time of 1 minute or longer and had 739 go unanswered. These are areas we need to focus on in conducting training with employees to address these issues. We can provide better customer service and these results give us the opportunity to establish reliable benchmarks that we can track and use in annual evaluations and in developing focused training.

These are just examples of some of the benefits available within the results we were able to get from the text classification model we developed for our chat interactions.

Future Work

Conclusion

Bibliography

Whitchurch, Michael J., and Erin Merrill. “Chat Response Competency: Library Professionals vs. Undergraduate Student Employees.” JOURNAL OF LIBRARY AND INFORMATION SCIENCES, vol. 7, no. 2, 2019.http://jlisnet.com/journals/jlis/Vol_7_No_2_December_2019/2.pdf

Mitchell 1997 http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.html

Appendix I: Custom Transformer Process

Custom Transformer Tokenizer Class:

```
class TransformersBaseTokenizer(BaseTokenizer):
    """Wrapper around PreTrainedTokenizer to be compatible with fast.ai"""
    def __init__(self, pretrained_tokenizer: PreTrainedTokenizer,

                 model_type = 'bert', **kwargs):
        self._pretrained_tokenizer = pretrained_tokenizer
        self.max_seq_len = pretrained_tokenizer.max_len
        self.model_type = model_type

    def __call__(self, *args, **kwargs):
        return self

    def tokenizer(self, t:str) -> List[str]:
        """Limits the maximum sequence length and add the special tokens"""
        CLS = self._pretrained_tokenizer.cls_token
        SEP = self._pretrained_tokenizer.sep_token
        if self.model_type in ['roberta']:
            tokens = self._pretrained_tokenizer.tokenize(t,
add_prefix_space=True)[:self.max_seq_len - 2]
        else:
            tokens = self._pretrained_tokenizer.tokenize(t)[:self.max_seq_len - 2]
```

```
        return [CLS] + tokens + [SEP]
```

The Basic DataBunch API from FastAI, this is the portion that we did:

```
databunch = (TextList.from_df(train, cols='Phrase', processor=transformer_processor)
            .split_by_rand_pct(0.1,seed=seed)
            .label_from_df(cols= 'Sentiment')
            .add_test(test)
            .databunch(bs=bs, pad_first=pad_first, pad_idx=pad_idx))
```

Custom Transformer Wrapper Class:

```
# defining our model architecture
class CustomTransformerModel(nn.Module):
    def __init__(self, transformer_model: PreTrainedModel):
        super(CustomTransformerModel,self).__init__()
        self.transformer = transformer_model

    def forward(self, input_ids, attention_mask=None):
        logits = self.transformer(input_ids,
                                  attention_mask = attention_mask)[0]
        return logits
```

Transformer Config for our Model:

```
config = config_class.from_pretrained(pretrained_model_name)
config.num_labels = 5
```

The FastAI Learner API that combines everything we've created together:

```
learner = Learner(databunch,
                  custom_transformer_model,
```

```
                opt_func = CustomAdamW,
                metrics=[accuracy, error_rate])
```

Training Loop:

```
learner.save('untrain')
learner.load('untrain');
learner.freeze_to(-1)
learner.lr_find()
learner.fit_one_cycle(1,max_lr=2e-03,moms=(0.8,0.7))
learner.save('first_cycle')
learner.load('first_cycle');
learner.freeze_to(-2)
lr = 1e-5
learner.fit_one_cycle(1, max_lr=slice(lr*0.95**num_groups, lr), moms=(0.8, 0.9))
learner.save('second_cycle')
learner.load('second_cycle');
learner.freeze_to(-3)
learner.fit_one_cycle(1, max_lr=slice(lr*0.95**num_groups, lr), moms=(0.8, 0.9))
learner.save('third_cycle')
learner.load('third_cycle');
learner.unfreeze()
learner.fit_one_cycle(2, max_lr=slice(lr*0.95**num_groups, lr), moms=(0.8, 0.9))
```

Appendix II: Custom Transformer Model Architecture

```
CustomTransformerModel(
  (transformer): DistilBertForSequenceClassification(
    (distilbert): DistilBertModel(
      (embeddings): Embeddings(
        (word_embeddings): Embedding(30522, 768, padding_idx=0)
        (position_embeddings): Embedding(512, 768)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
      )
      (transformer): Transformer(
        (layer): ModuleList(
          (0): TransformerBlock(
```

```
    (dropout): Dropout(p=0.1, inplace=False)
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (1): TransformerBlock(
    (dropout): Dropout(p=0.1, inplace=False)
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (2): TransformerBlock(
    (dropout): Dropout(p=0.1, inplace=False)
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (3): TransformerBlock(
    (dropout): Dropout(p=0.1, inplace=False)
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (4): TransformerBlock(
    (dropout): Dropout(p=0.1, inplace=False)
    (attention): MultiHeadSelfAttention(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (ffn): FFN(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (5): TransformerBlock(
    (dropout): Dropout(p=0.1, inplace=False)
```

```
          (attention): MultiHeadSelfAttention(
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=5, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
 )
)
```

Appendix III: Methodology

Based on the Universal Language Model Fine-Tuning for Text Classification (ULMFiT) method of transfer learning in NLP, we have achieved a state-of-the-art system that can still be visibly improved. ULMFiT was developed by Jeremy Howard, a co-founder of fast.ai, which he uses to demonstrate state-of-the-art results using models trained on 100x less data than the originals, making it a very flexible and powerful algorithm. Some of the key techniques involved in ULMFiT namely Discriminative Learning Rates, Gradual Unfreezing, and Slanted Triangular Learning Rates have been essential to the reported level of results, and will be subsequently explained.

The network architecture utilized is a standard DistilBERT layered neural network as pioneered by HuggingFace🤗 NLP. We wrote a custom transformer wrapper to make it more flexible for testing and development, allowing us to substitute different base transformer language models (such as XLNet and RoBERTa) in the easiest way possible.Throughout this testing we learned that in order to optimize our network for the shortest amount of time, DistilBERT was the best choice, because it is the smallest. The network is initialized in layers in the following sequence:

1 - Pre-trained embedding layer

2 - Distilbert Transformer layer 1

3 - Distilbert Transformer layer 2

4 - Distilbert Transformer layer 3

5 - Distilbert Transformer layer 4

6 - Distilbert Transformer layer 5

7 - Distilbert Transformer layer 6

8 - Pre-Classifier layer

9 - Classifier

So on to what we've done differently. Discriminative Learning Rates propose a theory that to improve a model, a different learning rate should be used for different layers, with the highest learning rate at the very end of the model (pre-classifier layer). When learning with Stochastic Gradient Descent, the equation changes slightly:

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta)$$

Stochastic Gradient Descent (SGD). η is learning rate while ∇θJ(θ) is the gradient of objective function. (Howard and Ruder, 2018)

$$\theta_t^l = \theta_{t-1}^l - \eta^l \cdot \nabla_{\theta^l} J(\theta)$$

SGD with discriminate fine-tuning. η l is learning rate of l-th layer. (Howard and Ruder, 2018)

Slanted Triangular Learning Rates build off of the first idea by increasing the dynamic learning rate at the beginning of training, then decaying it linearly to form a sort of triangle graphically:

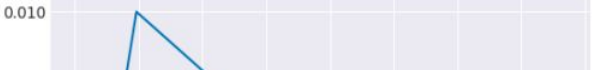
$$cut = \lfloor T \cdot cut\_frac \rfloor$$

$$p = \begin{cases} t/cut, & \text{if } t < cut \\ 1 - \frac{t - cut}{cut \cdot (1/cut\_frac - 1)}, & \text{otherwise} \end{cases}$$

$$\eta_t = \eta_{max} \cdot \frac{1 + p \cdot (ratio - 1)}{ratio}$$

STLR Formula. T is number of training iteration. cut_frac is the fraction of increasing learning rate. cut is the iteration switching from increasing to decreasing. p is the fraction of the number of iterations which increase or decreased. (Howard and Ruder, 2018)

Gradual Unfreezing bring the first two ideas home, by starting the training with the entire model frozen except for the very last layer (thus having the highest learning rate)



during the first epoch of training, then drop the learning rate slightly and do the second round of training with everything frozen but the last 2 layers, then the last 3 layers, then with the lowest learning rate, we unfreeze the entire model and train for 2-5 epochs, with the learning rate decreasing each time.

This basic methodology has allowed us to achieve state-of-the-art results on 21 categories, with only 4475 labelled examples in our training set. This dataset is relatively small, making this achievement that much more impressive. With that in mind, our results could be much more useful for the purposes of BYU Library if we put some more effort into cleaning our current dataset, augmenting outlier examples, and add more data into it.

Appendix IV: Graphs