Dev Technology Group Description of Methods for GSA Artificial Intelligence (AI) Machine Learning (ML) EULA Challenge 2020

1. Contact information for Official Representative:

Name: Joshua Powers
Email: joshua.powers@devtechnology.com
Team Name: Dev Technology

2. Names of additional team members:

Name: Michael Oduyebo
Name: Niroop Gonchar
Name: Zach Lawrence
Name: Sherri Elliott
Name: John Janek

3. Dev Technology EULA Clause Analysis Architecture:

a. Technology Scope:

Our technical submission is implemented in Python 3.7. We created a module, **dteulaml**, for exploring features, training models and evaluating hyperparameters. We created a separate module, **dteulaapp**, for implementing the single-page web application and clause analysis API. The API is responsible for converting PDF or Microsoft Word (docx) text to plain text; extracting clauses from plain text; classifying clauses as acceptable or unacceptable; and finding closest matches between classified clauses and known acceptable and unacceptable clauses. The single-page and the API are served as a Flask application.

Because we host an instance of these resources in public on an AWS EC2 at our lab's website (https://eulacheck.devlab-dtg.com), we front them with an SSL-enabled Apache server using reverse proxy to load balance the backend API. This is the recommended posture for exposing the app as a public-facing endpoint. These resources can be served locally using just Flask at http://localhost:5000 by using the command-line interface (CLI):

```
python -m dteulaapp --operation app
```

Specific Python dependencies used include:

Python Dependency	Description
Flask 1.1.2	https://flask.palletsprojects.com/en/1.1.x/ is Python's basic application and web server. We use it to serve both our single page application and our analytic API.

Keras 2.4.3	https://keras.io/ is the Python Deep Learning framework that wraps Tensorflow, Google's Apache-licensed open source deep learning platform. We do not use Keras directly, but it is required for the Simpletransformers module that we do interact with.
Numpy 1.19.1	https://numpy.org/ is the standard numerical calculation library for Python.
NLTK 3.5	https://www.nltk.org/ is a widely-used NLP library. We use it for feature exploration.
Pandas 1.0.5	https://pandas.pydata.org/ is the standard machine learning dataset manipulation library for Python.
PyPDF2 1.26.0	https://pythonhosted.org/PyPDF2/ is the library we use for parsing text from PDF documents.
Python-docx 0.8.10	https://python-docx.readthedocs.io/en/latest/ is the library we use for parsing text from Microsoft Word (docx) documents.
PyTorch 1.3.1	https://pytorch.org/ is another deep learning framework, developed by Facebook, and is required for some numeric calculations in the Simpletransformers module.
Scikit-learn 0.23.1	https://scikit-learn.org/stable/ is a widely used machine learning API. We use their implementation of metrics to measure the fitness of the trained models.
Scipy 1.5.1	https://www.scipy.org/ includes numerical computations that are not part of Numpy
Simpletransformers 0.45.0	https://simpletransformers.ai/ is an NLP library that allows rapid development, evaluation and exploration of BERT and other transformer-based deep learning models as they apply to your specific problem. It offers a streamlined approach to rapid prototyping that we were able to use in this challenge.
TensorboardX 2.1	https://pypi.org/project/tensorboardX/ is a library that offers various visualizations of Tensorflow processes as they run.
Transformers 3.0.2	https://huggingface.co/transformers/ is a wrapper around PyTorch and Tensorflow for building and using transformer-based deep learning models such as BERT. We do not use Transformers directly, but it is required for the Simpletransformers module that we do interact with.

b. Functionality and User Interface:

Our single page web application is demonstrated fully in our Solution Demonstration submission. It is responsible for the following:

- Allowing a user to upload a PDF or Microsoft Word (docx) file with EULA content in it
- Or, allowing a user to paste plain text from a EULA document into a resizable text box
- Allowing a user to submit either of these items to analysis by our classifier and nearest match engine

- Showing the user a table of all clauses found in their submission, along with the following:
    - A determination of acceptability with respect to Federal regulations (Acceptable, Unacceptable, Not Sure) with a confidence score (Probability of Acceptable)
    - A set of radio buttons for the user to provide feedback for each clause's result.
    - The nearest match to each clause from a library of known acceptable clauses.
    - The nearest match to each clause from a library of known unacceptable clauses.

The back-end API, also exposed as a Flask endpoint, is responsible for the following:

- Converting PDF or Word to plain text
- Extracting clauses from the plain text. We use different techniques depending on the original format of the EULA, as there are differences in the nature of the extracted text.
- Classifying each clause as Acceptable, Unacceptable or Not Sure
- Scoring the probability that the clause is Acceptable
- Finding the closest Acceptable clause from the set of known Acceptable clauses
- Finding the closest Unacceptable clause from the set of known Unacceptable clauses

There is also a CLI for the dteulaapp module for processing files in batch locally:

```
python -m dteulaapp --operation batch --input /home/ubuntu/docs
--output /home/ubuntu/results.json
```

Results of batch processing will be written to the designated output file as a JSON list of objects with attributes for source file, original text, classification (Acceptable, Unacceptable, Not Sure), confidence (0.0-1.0 that the clause is acceptable), closest known acceptable match, closest known unacceptable match.

c. Application of Artificial Intelligence/Machine Learning (AI/ML):

Overall Approach

Dev Technology first analyzed the nature of the challenge problem, the dataset and the features found within the dataset. The results of this analysis led us to choose a classification approach based on a language model derived from a much larger corpus than the clause training set. The most comprehensive language model in use in industry and academia is the Bidirectional Encoder Representations from Transformers (BERT) model. This model is built from the Wikipedia and Google Books corpora. It allows us to transfer learning from these large datasets to the clause acceptability problem. We chose train/test sets from the original training data and explored the hyperparameters available to a deep learning approach using a classification layer on top of the base BERT neural network. Once we settled on a set of parameters, we trained a final model on selected training data and evaluated it on the full dataset provided by the GSA. We then used that model to prepare predictions for the validation data, also provided by GSA.

Dataset Exploration

The training dataset provided by GSA included 7879 clauses. Each had an integer identifier, some amount of UTF-8 encoded English text and a classification of '1' for unacceptable or '0' for acceptable. Text sizes ranged from a single character to 46,274 characters. Clause ID 133 was completely empty of text, for instance, containing a single newline character. Clause ID 3696, on the other hand, was over 7000 words long, and appeared to be an entire EULA document. We limited training data to clauses with between 10 and 500 words to avoid distracting artifacts. The results of this analysis are listed in our submission as 'textanalysis.csv' in our Input Data folder.

Also included in the GSA package was a set of labeled token lists from a previous machine learning effort involving Section 508 compliance analysis. While the language was similar in some ways to the EULA clause content, the specific learned parameters were not present in the provided model, so the opportunity for transfer learning would require that tokenization methods were identical to the 508 model and that we could reproduce an acceptable classification accuracy from those token lists. This restriction led us to seek a transfer learning approach from the BERT project, as documented below.

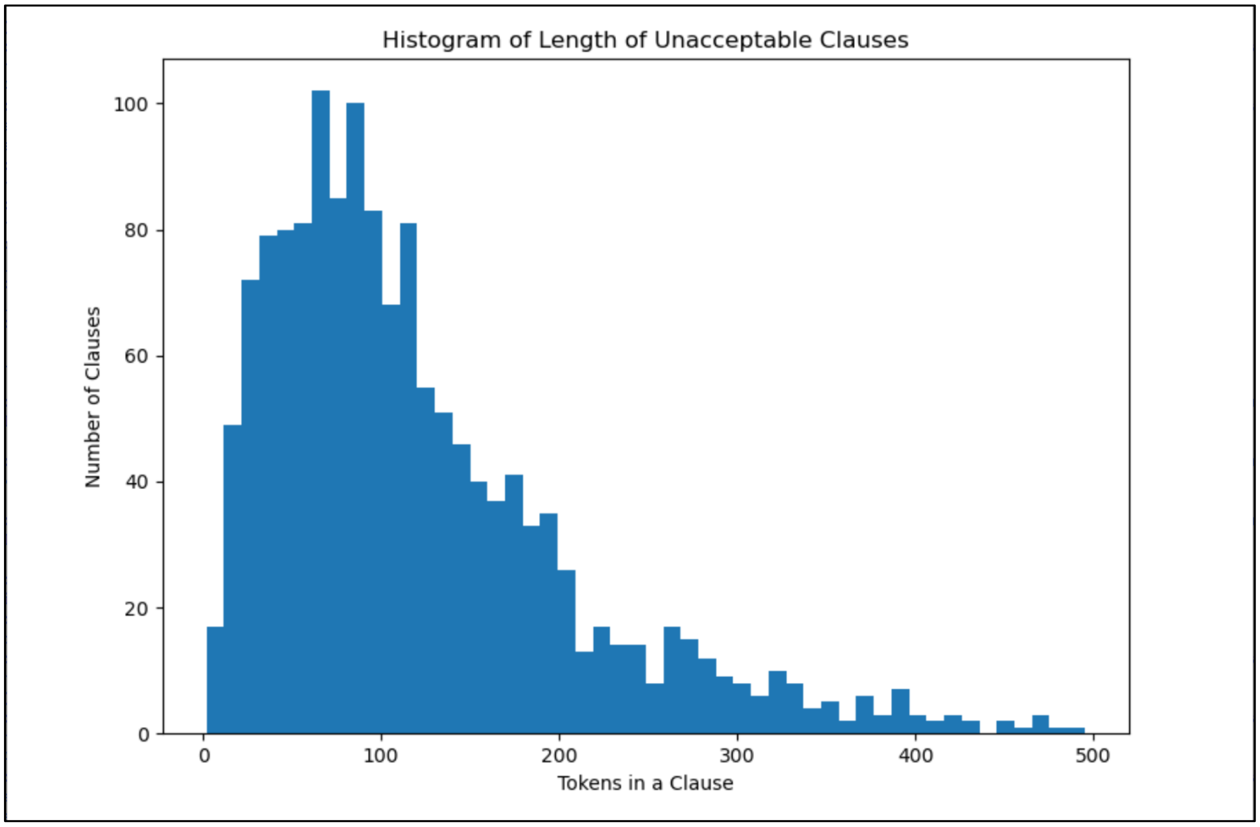

Figure 1 - Unacceptable Clauses Tended to be Between 30 and 200 Tokens Long

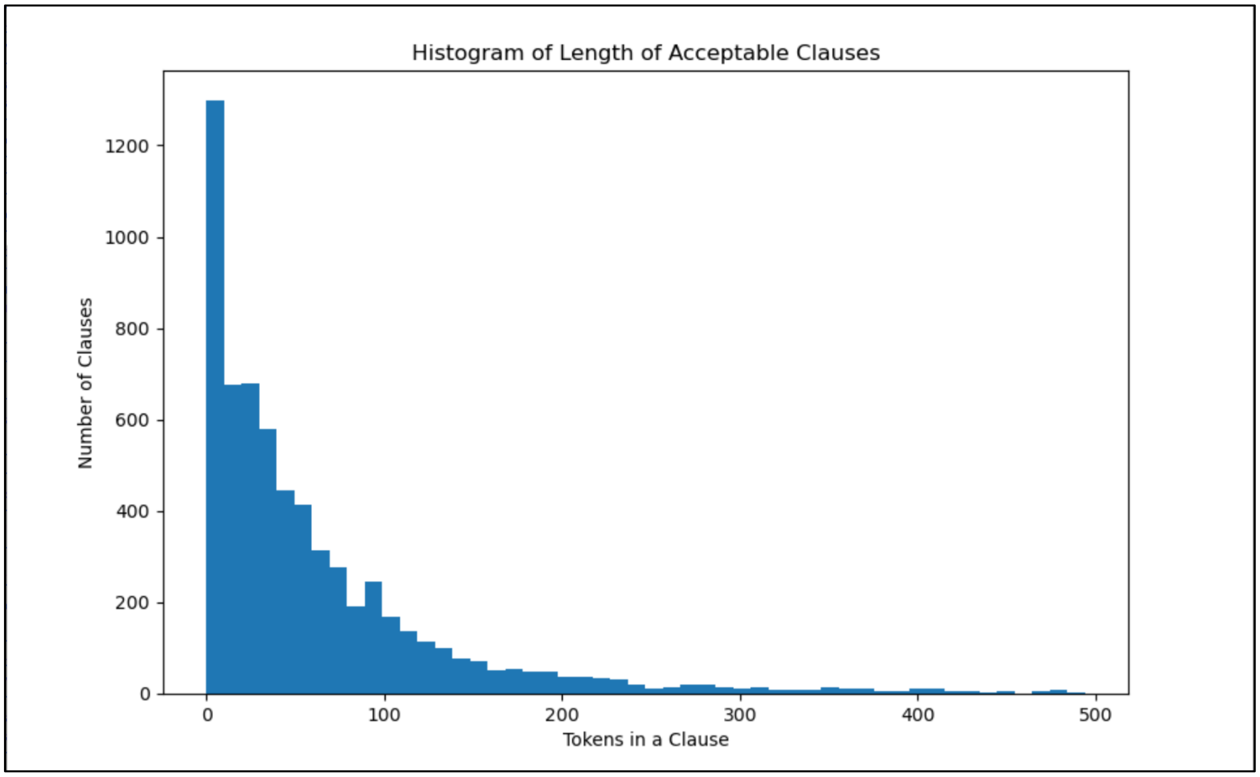

Figure 2 - Acceptable Clauses Were Much More Likely to be Short

We identified 45 pairs of clauses which were exact duplicates of each other, but which were given different labels in the training data. An example of such a pair are Acceptable Clause 2063 and Unacceptable Clause 2064, the texts of which follow:

2063: "3) Liquidated Damages FAR 52.219-16: In accordance with its established business practice, Company will provide its corporate small business subcontracting plan rather than a specific small business subcontracting plan for its GSA Schedule Contract. Company's plan has been approved by DISA. Since Company performance against this plan is not directly linked to its GSA Schedule Contract, Company takes exception to the inclusion of FAR 52.219-16, Liquidated Damages – Subcontracting Plan in its GSA Schedule Contract"

2064: "3) Liquidated Damages FAR 52.219-16: In accordance with its established business practice, COMPANY will provide its corporate small business subcontracting plan rather than a specific small business subcontracting plan for its GSA Schedule Contract. COMPANY's plan has been approved by DISA. Since COMPANY performance against this plan is not directly linked to its GSA Schedule Contract, COMPANY takes exception to the inclusion of FAR 52.219-16, Liquidated Damages – Subcontracting Plan in its GSA Schedule Contract"

We used Dice similarity scores [Dice (1945)] to calculate such potential duplicate clauses between the label populations using 1-grams, 2-grams and 3-grams. We found 503 pairs with a Dice score of over 0.8. These findings are listed in our submission as 'conflicts.csv' in our Input Data folder. When we limited training data to clauses which did not have any matches of 0.95 or higher with a clause in the other label population, performance did not improve. It is possible that having these conflicting labelings in the training set helped avoid overfitting. In testing the complete dataset, however, there is no way to avoid at least 45 bad results.

The dataset included 1472 examples of unacceptable clauses, and 6407 examples of acceptable clauses. This imbalance would have implications for hyperparameters chosen in the final model. There is potential future work described later which could automatically augment the training set.

Feature Exploration

Since this is a natural language classification task, we explored the vocabulary of the clauses. We identified unique 1-grams, 2-grams and 3-grams, but did not allow n-grams to cross punctuation boundaries. This resulted in a vocabulary of 114,800 n-grams. We then calculated the Bayesian probability that an n-gram's presence would indicate an unacceptable clause. These results are listed in our submission as 'ngrampred.csv' in our Input Data folder.

We used the GSA-provided document 'appendix_b_unacceptable_clauses.pdf' to attempt to link distinctive words and phrases to acceptability classifications. For instance, unacceptable class 2 is described as:

"Contract formation via using, downloading, clicking "I Agree," etc. (commonly known as shrinkwrap/clickwrap/browsewrap), or a provision purporting to bind the Government to a set of terms posted at a specified URL"

An exploration of n-grams including words such as 'click' and 'download' yields the following probabilities that clauses are acceptable:

N-gram	Clauses	Probability Acceptable
download	87	70%
downloading	28	35%
to download	24	75%
clicking	18	5%
downloaded	16	93%
download or	16	81%
by clicking	14	7%

This analysis led us to attempt a classifier based on the presence of low-probability acceptable vocabulary. This classifier is implemented as 'AggregateBayes' in our submission source code Python module dteulaml. This was a quick and dirty modification of Naïve Bayes to use n-grams and account for clause length differences. It did not yield very good results, but future work on it may improve. It is an attractive approach because the specific objectionable n-grams of an unacceptable clause can be easily identified. It tends to overfit, however, and requires exact term matches to work properly. Because it is non-parametric, typical approaches to address overfitting, such as regularization, are not available.

Predictive Model

Since the BERT project first published their results in 2018, there has been a lot of activity in NLP using their trained models as a starting point for classifiers, text generation and semantic search. The Python library SimpleTransformers packages the pre-trained BERT models and variants in a way that makes it fairly straightforward to extend with binary and multi-classification datasets. We used the Robustly Optimized BERT model (RoBERTa) [Liu et al.(2019)] which is an upgrade on the original BERT model.

Bidirectional encoder representation transformers are self-training deep learning neural networks which assign themselves the task of predicting masked keywords given surrounding (both prior and later) keywords. The final hidden layer of these networks is used to represent each word in the vocabulary as a series of weights (768 weights in the case of the RoBERTa model we used). These 'word vectors' can then be transferred to other machine learning tasks, and encode the sum semantic knowledge about the words from all the BERT training that was performed to create them.

Using this form of encoding to classify potentially very long texts requires passing a fixed-size window over the text and developing a classification for that window. The traditional method for whole text classification with windows is to take the mode of the window labels as the classification of the entire text. Because of the zero-tolerance nature of clause unacceptability, in theory any single window that has an unacceptable label should cause the entire clause to be unacceptable. In practice, when we implemented this strategy, we did not see an improvement in classification.

The final model used in the preparation of the validation data results is included in our submission under the DevTechnology_Compiled_Models folder as 'eulabert'. There are several components of the model in the folder, including a vocabulary (vocab.json), trained parameters (pytorch_model.bin), hyperparameters (model_args.json) and neural network configuration (config.json).

Hyperparameter Exploration

We explored a variety of hyperparameters to understand the best training scenario for our model. We primarily used F1 measure and Matthews Correlation Coefficient (MCC) to measure the suitability of each parameterization. For each combination of parameters, we split the training data into an 80% training set and a 20% test set. Our metrics were based on evaluation of each model on the test set. Specific parameters we explored included:

- *Epochs*: How many rounds of training before considering the model finished. Too few epochs risks bias, and too many overfitting. We explored 1-8 epochs.

- *Early Stopping*: If, during training, the metrics do not improve by a certain amount after an epoch, allow the training to halt early and declare the best model from an earlier epoch. We explored using this technique.

- *Label Weights*: to account for the imbalance of the dataset (19% unacceptable), we can weight the influence the label has during training. We explored weighting unacceptable labels 0.5, 1, 2, and 3.

- *Window size*: Because the inputs are of variable size, we must pass a window over the text. We explored window sizes of 32, 64, 128 and 256 tokens.

- *Stride*: When we pass the window over the text, how much overlap should the next window include from the former window? We explored strides of 0.7, 0.8 and 0.9, meaning the percent of a window size that moves 'forward' for the next window.

- *Include Conflicts*: Should we include conflicting texts (labeled both acceptable and unacceptable) in the training set? We briefly explored using conflict-free data.

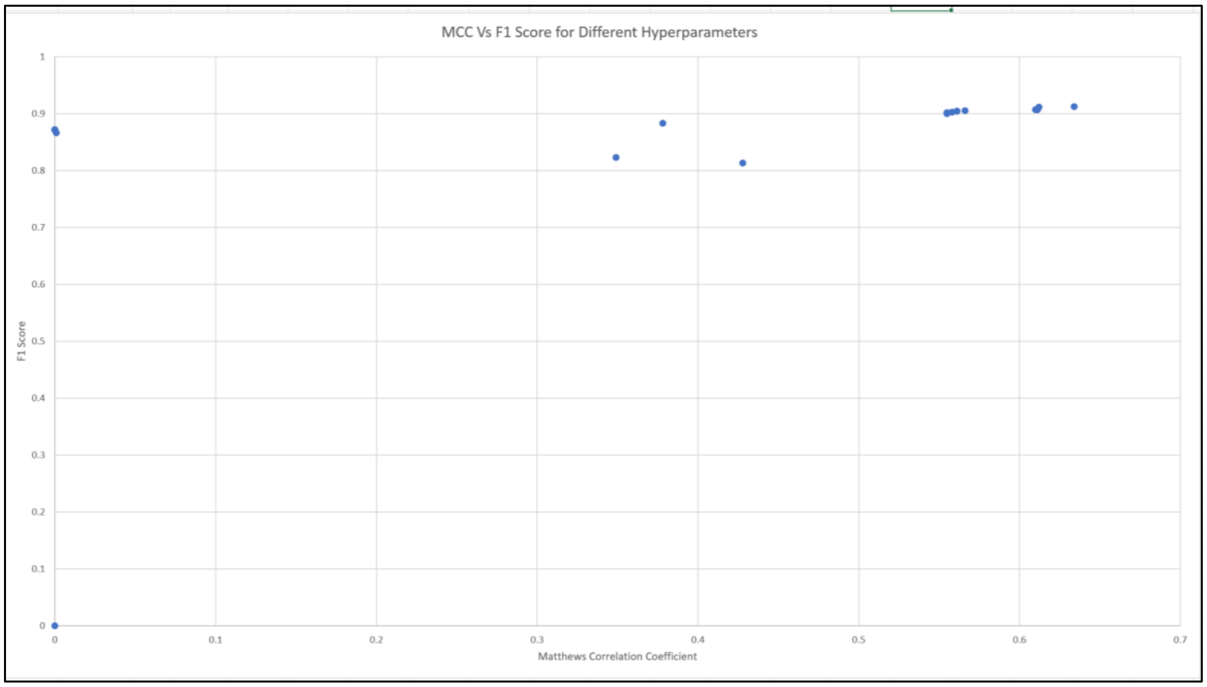

Figure 3 - Matthews Correlation Coefficient and F1 Scores for BERT Model Hyperparameter Configurations

There is potential for catastrophic results caused by the classifier always choosing 'acceptable' or always choosing 'unacceptable.' We did not identify a common theme to parameterizations that seemed to lead to these cases, but it bears further exploration.

The F1 and MCC scores for all these parameterizations are listed in our submission as 'hyperparameters.xlsx' in the DevTechnology_Code_and_Data folder.

Metrics

The final model we built used the best combinations of parameters that we found in the above described exploration. These were:

- Epochs: 5
- Early Stopping: No
- Label Weights: 2, 1
- Window size 256 tokens
- Stride: 0.9
- Include Conflicts: Yes

During hyperparameter testing, on the 20% hold-out test data, this model had 0.93 precision and 0.9 recall.

For the final submission, we trained the model using these parameters on all training data between 10 and 500 tokens in length.
We then evaluated it on the complete training data to obtain our self-reported metrics. The following are the metrics obtained:

- Precision: 0.89
- Recall: 0.94
- F1 Score: 0.91
- Brier Loss: 0.034
- Matthews Correlation Coefficient: 0.89

Precision / Recall Curve:

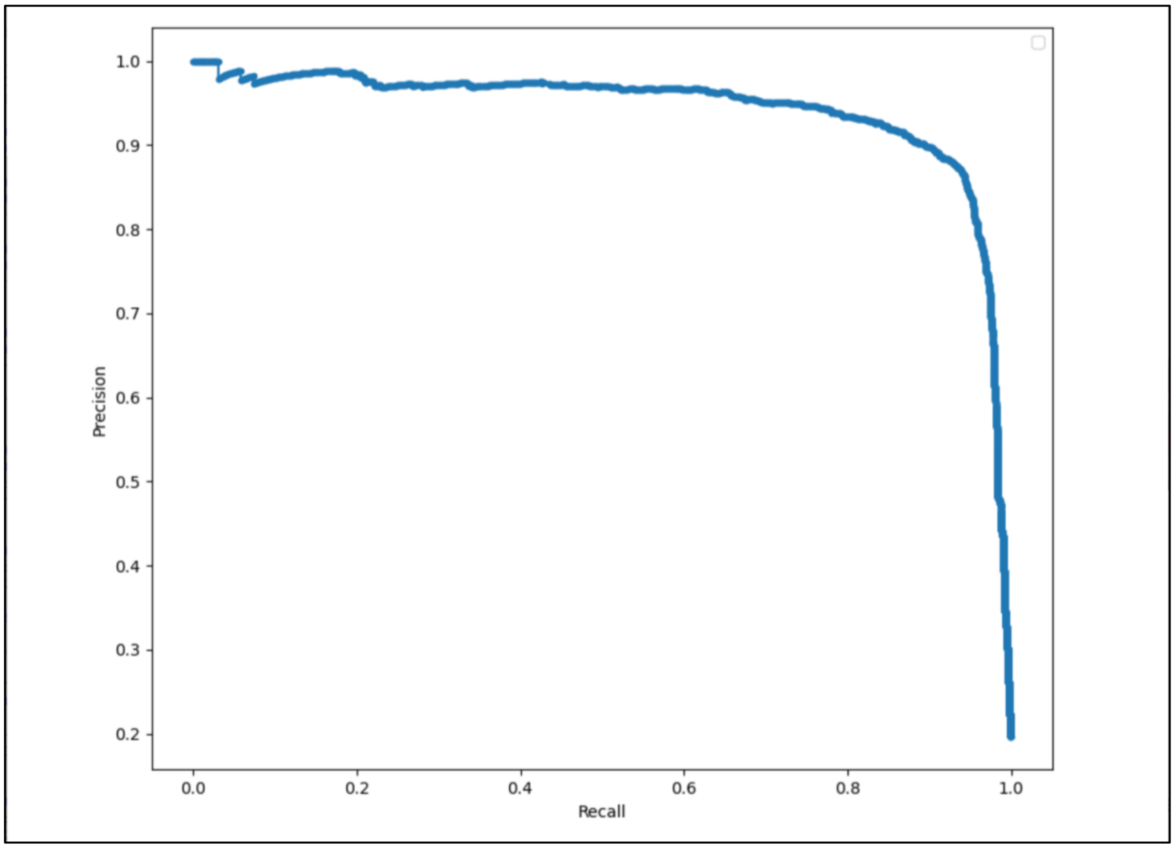

Figure 4 - Precision / Recall Curve for BERT-based EULA Clause Classification

Additional Functionality

In addition to the classification of clauses, we built a clause extractor that identifies clause boundaries in plain text. This extractor uses a combination of regular expressions and finite state machines to identify the context of a particular line of text and decide if it represents a clause boundary. This is used in the API that drives the single-page app.

We also used Dice similarity to search the training clauses for acceptable and unacceptable clauses that best match novel input clauses. Again, this is to support the single-page app and to provide some evidence of why a clause might have been classified the way it was. While pointing to specific regions of text did not make it into our demonstration, the ability to see similar, known texts gives the user insight into why classifications were made.

Future Work

- *Dataset Imbalance*: The imbalanced nature of the training dataset could be addressed by using a machine translation service such as Google's Neural Machine Translation to automatically translate clauses from the minority label into another language, and then back into English. This would provide different linguistic utterances which would be likely to still carry the underlying meanings of the clauses. Depending on scale, this could be somewhat costly, but would only have to be executed once to supplement the dataset.

- *Classes*: If the listing of reasons for clauses being unacceptable is (relatively) complete in the GSA-provided document 'appendix_b_unacceptable_clauses.pdf', this could be extended to a multi-classification problem using each listed description as a distinct class. Clauses could then be classified into one of 15 classes, and much more specific reasons could be given for a clause's labeling as unacceptable. This would obviously require more manual labeling effort, but using a bootstrapping approach with a user interface to guide labelers could speed the process up.

- *Windowing*: If we continue with an encoder-based classification system, it may be beneficial to identify specifically which window of text in an unacceptable clause makes it so, and train on smaller windows of 'completely' unacceptable content. Classification could then be performed as finding any unacceptable window in the text. This would also require more manual labeling effort which could be bootstrapped with an online learning algorithm and a browser-based feedback app.

- *Transfer Models*: There are many BERT-like models of natural language in the machine learning community. Some are trained with case sensitivity, on vertical corpora or with multilingual content. We would gain more insight about the language model by exploring EULA classification performance using these different models.

4. Citations:

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (https://arxiv.org/abs/1810.04805)

Dice, Lee R. (1945). "Measures of the Amount of Ecologic Association Between Species". Ecology.

Liu, Y. and 9 colleagues 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. (https://arxiv.org/abs/1907.11692)