Submission for GSA AI/ML EULA Challenge 2020

Contact information:

Name: Rohan Narain
Email: narain.rohan@berkeley.edu
Team Name: AnNIHilators

Names of additional team members:

Name: Andrew Kim, andrew.kim@nih.gov

Introduction to team:

Rohan is a recent graduate from UC Berkeley, where he studied Statistics and Data Science. Rohan just finished up the Civic Digital Fellowship program at NIH, where he used transformer models to extract information for the categorization of NIH grant applications in the Division of Scientific Categorization and Analysis.

Andrew is a front end engineer who currently works at the National Center for Biotechnology Information on PubMed labs. He has a wealth of experience designing simple, intuitive UIs and user experiences for all kinds of applications.

Executive summary of solution:

The solution consists of a web application where users can upload EULA documents in either .docx or .pdf format and view individual clauses along with predictions for whether or not they are acceptable under federal contractor regulations. The document gets sent to an AWS endpoint which parses the document and classifies each individual clause in the contract. The classifier was trained on 6,698 EULA clauses from the GSA.

Classifier and API Architecture:

A. Technology Scope

- This solution uses services in AWS, orchestrated by a Cloud Formation stack formed by the open-source tool Cortex. Cortex creates an API Gateway and Elastic Container Service for Kubernetes (EKS) cluster in AWS which allows an application to serve a model that runs predictions on data passed from HTTPS requests to the API Gateway. Further, the solution uses a model stored in an AWS Simple Storage Service (S3) bucket.
- Through Cortex, we requisitioned an Elastic Compute Cloud (EC2) instance with NVIDIA GPU (specifically the g4dn.xlarge instance). This sped up inference almost 100 times for sample documents compared to regular CPU.

- To train the model, we used Google Colab, a service provided by Google in which users can run Jupyter Notebooks for data science and machine learning tasks using GPU hardware acceleration.
- We used a pre-trained DistilBERT model from the Python package ktrain. This package provides a low-code machine learning and AI workflow for training, evaluating, and deploying models in production. A demonstration of how this was used is included in the source code in “train_distilbert.ipynb”.

B. **Functionality and UI**

- The solution uses an application in which users can upload multiple PDF or DOCX files and receive a display of the contract along with predictions for acceptability of each clause.
- The web app is built on the Vue.js framework and utilizes Vuetify for material design components (popover, tabs, file input, circular progress, .etc). We also utilized larsjung’s pagemap mini map package to help users understand at a glance where the clauses of interest lie in the context of the whole document.
- For the front-end, we have used Netlify to deploy the static build files which are stored on Github.
- Users can also analyze the probability that each clause is acceptable and even send feedback to relabel the clauses if they think the model misclassified a clause. These relabels are stored in a Google Cloud Firestore NoSQL database.

C. **Application of AI/ML**

- The application uses a distilled version of BERT, a popular, state-of-the-art neural language model for text classification. Before being released publicly, BERT was pre-trained on a very large text corpus from across the internet. HuggingFace released a PyTorch implementation of this model for public use. The HuggingFace model can be fine-tuned to whichever task and dataset one is using--thus, this is a **transfer learning method**.
- BERT is a very large model, so using it for inference can be slow. For this reason, we elected to use DistilBERT, HuggingFace’s distilled version of BERT which has 60% fewer parameters but retains 95% of the regular BERT’s performance. DistilBERT still maintains this level of performance because of the amount of text the original BERT was trained on.
- We fine-tuned a DistilBERT model on the GSA training data given to us. We set aside 10% of the given data (788 clauses) as test data. When a user submits a file to our app, our fine-tuned DistilBERT, served from an S3 bucket, runs inference on the parsed document and returns predictions for individual clauses along with the probabilities that the clauses are acceptable.

Performance Metrics

Note: When splitting the data in training and test sets, we used the seed 123 *and scikit-learn’s train_test_split function. In addition, the F1-score was taken with a weighted average to account for class imbalance and produce a score more representative of the model’s performance.*

First, we'll derive a benchmark using a "baseline model", which predicts the class 0, or "acceptable", in every case. This gives an accuracy score of 0.813. This also gives the following performance metrics:

- F1 score: 0.0 (by definition)
- Brier score: 0.545 (labels compared to probability of picking 0 each time)

Our DistilBERT model performed as follows on a test set of size 788:

- Brier score loss: 0.098
- F1 score: 0.870
- Additional metrics:
    - Accuracy: 0.871
    - Precision: 0.870

Validation Data File

For the validation data, we ran each of the clauses through our compiled DistilBERT model using ktrain. The model used the relationships it extracted from the training data to provide predictions for each of the clauses in the validation file.

Compiled Models

The compiled DistilBERT model is not available in the solution because it is too large for Github's file size limit.