# **Contact information for Official Representative:**

**Name: Charles McAllister**

**Email: mcallistercs@gmail.com**

**Team Name: Charles McAllister**

# **Names of additional team members:**

**N/A**

# **Introduction to Team:**

A Returned Peace Corps Volunteer (Ethiopia 2012-14) with a Master of Public and International Affairs, I'm currently a data scientist in the AI and Engineering division of KPMG’s Digital Lighthouse, consulting Federal agencies on how to apply machine learning and natural language processing to their missions.

Before turning to data science, I held a variety of positions across the private, public, and non-profit sectors. I've furthered the cause of nursing home reform through data analysis at the National Consumer Voice; conducted market research to support small businesses on their export strategies with the U.S. Commercial Service; taught English overseas while in the Peace Corps; counseled nonprofit organizations on grant writing during graduate school; and co-developed a web application that uses machine learning to monitor Federal IT procurements' compliance with Federal law at the General Services Administration’s Office of Government-wide Policy.

# **Executive Summary of Solution:**

Binarization of End User License Agreements Recommendations, BEULAR for short, is a dockerized web-application, built with Python Flask using Redis as a service worker, that utilizes a suite of optional AWS services on the back-end to enable the machine-learning lifecycle, including model building, training, deployment and, most importantly, re-training and re-deployment in response to inferences that humans have validated on the front-end via the web application.

From a user’s perspective, the app:

* allows users to upload one or more Word or PDF document(s)
* parses uploaded documents into their constituent clauses without expensive API calls
* presents the predictions using color-coding to call out potentially non-compliant clauses
* allows users to see an explanation for each prediction, showing them the prediction’s probability score as well as the influence of the top 10 words/phrases in the clause toward generating that prediction
* allows users to validate predictions, with an option for providing open-ended feedback, and submits that data to a cloud-hosted datastore where the models can be re-trained and therefore incrementally improve over time

For government data scientists, the submission also includes an AWS Cloud Developer Kit application that provisions all of the cloud-based machine learning and API infrastructure using infrastructure as code. This gives the government access to the same elastic cloud resources that I used to build, train and deploy models.

# **BEULAR Architecture:**

## **Technology Scope:**

* BEULAR (Web Application)
  + Python Flask (web application framework)
  + Docker (dependency management)
  + Heroku (web hosting)
  + Redis (service worker)
  + Bootstrap (CSS)
  + Jquery (client-side javascript)
* BEULAR-API (Data Science Backend)
  + AWS Cloud Developer Kit (infrastructure-as-code)
  + AWS SageMaker (model development and hosting)
    - hosted in the private subnet of an AWS Virtual Private Cloud instance
  + AWS Lambda (API proxy)
  + AWS API Gateway (REST API service)
  + AWS S3 (data storage)
* BEULAR-NB (Modeling source code0
  + Python (sklearn and sagemaker) for modeling
  + Python’s nltk library for NLP and transfer learning

## **Functionality and User Interface:**

* The user interface is a web application
* The web application supports PDF and Word (.doc and .docx) formats for input, although the following additional file formats could be supported with the inclusion of more dependencies in the docker image:
  + .csv
  + .eml
  + .epub
  + .json
  + .html and .htm
  + .msg
  + .odt
  + .pptx
  + .ps
  + .rtf
  + .tiff and .tif
  + .txt
  + .xlsx
  + .xls via xlrd
* One or more documents can be processed at a time.

## **Application of Artificial Intelligence/Machine Learning (AI/ML):**

* Since the challenge provided labeled data, I utilized supervised machine learning for an estimator. However, it’s important to note that this solution isn’t one algorithm: it’s a pipeline of algorithms which, together, generate predictions. Four separate model pipelines are provided with my submission.
* Feature Extraction
  + Term-Frequency Inverse-Document Frequency for the decision trees and linear models
    - Tokenization using Python’s NLTK library’s punkt model, which uses transfer learning to separate each clause into individual words (i.e. tokens)
    - For some experiments, I also used more transfer learning from NLTK’s wordnet module to stem or lemmatize each token
  + BlazingText Word-embeddings
    - Transfer learning used to create highly-dimensional vectors that represent the semantics of each word in a clause. These are fed to a neural network that makes probabilistic predictions.
* Feature Selection
  + Optional dimensionality reduction for the decision tree and linear models using truncated singular value decomposition to retain the data’s sparsity.
* Estimation
  + In order to fulfill the requirement for probabilistic predictions, I limited myself to probabilistic classifiers, such as logistic regression and decision-trees.
    - Blazing Text
      * Although this model is a neural network, it uses only one hidden layer for average pooling on the input word embeddings and then uses the softmax function to normalize that hidden layer’s vector *z* of *K* real numbers into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.
    - Decision Trees
      * The decision tree models learned from the data using a method known as bagging, where subsamples of the training data are used to build multiple decision trees, which are then merged together to get a more accurate and stable prediction. I used the following decision tree methods:
        + RandomForests

The RandomForests classifier builds multiple decision trees and splits nodes on the best split among a random subset of the features selected at every node using sampling with replacement.

* + - * + GradientBoosting

In GradientBoosting, each new decision tree is a fit on a modified version of the original data set, where predictions that were wrong are biased with increased feature weights. Predictions of the final ensemble model are the weighted sum of the predictions made by the previous tree models.

* + - Regression
      * Since labeled data was provided, I used stochastic gradient descent for the following regression-based models:
        + Logistic Regression

log loss function is used, which proves to be very efficient when the dataset has features that are linearly separable and tends to provide highly calibrated predicted probabilities

* + - * + Support Vector Machines

hinge loss function used, making the estimator less sensitive to outliers

Summary of Results

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Accuracy** | **F1** | **Brier Score** |
| **SGDClassifier** | **.79** | **.60** | **.15** |
| RFC | .78 | .56 | .15 |
| GBC | .83 | .49 | .15 |
| BlazingText | .84 | .53 | .12 |

The logistic regression model learned with Stochastic Gradient Descent performed best, so it is used by default with the web application. All of the other models are also included, however.