# **Contact information for Official Representative:**

**Name: Suresh Devalapalli**

**Email: sureshd@gaussiansolutions.com**

**Team Name: Gaussian Solutions**

# **Names of additional team members: None**

**Name:**

**Name:**

**Name:**

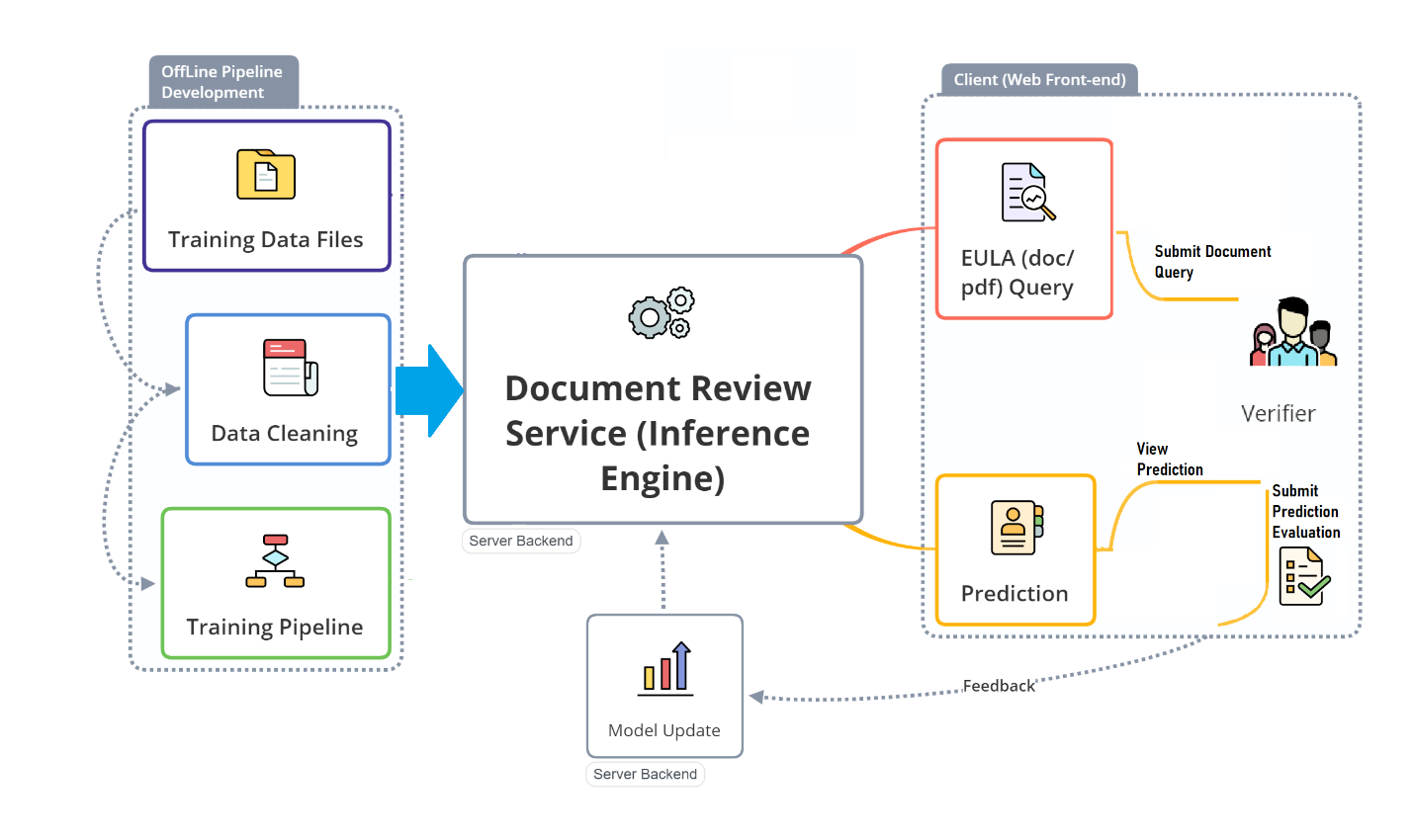
# **Introduction to Team:**

Gaussian Solutions LLC provides consulting services for clients seeking solutions in data analytics, AI and ML. Suresh Devalapalli is the sole proprietor of the company. Suresh has years of experience designing embedded systems, mainly for mobile phones, and gaming systems, and employing vision-based AI solutions. He is also a mentor at University of Washington CoMotion Labs that help faculty and students translate their research into viable products.

# **Executive Summary of Solution:**

Gaussian Solutions web-based solution allows a reviewer of EULA to upload the EULA document (in word, or pdf form) and receive whether each clause in the document is acceptable or not, and the confidence with which the recommendation is made. Given the confidence of the decision, reviewer can go over the less confident recommendations only, if he/she are pressed for time. Reviewer also has a chance to disagree with the recommendation. Reviewer’s changes to the recommendations are fed back to the server, and this data is used to retrain the model. Given the current sparsity of data, having a human assisted labeling, and data gathering will help build a better model over time. Gaussian Solutions also believes in continuous improvements, and continuous deployment (a.k.a CI/CD), whereby with each new feedback from reviewers, backend system retrains the model, measures the improvements, and automatically deploys the model, if it is better.

# **Gaussian Solutions Architecture:**



## **Technology Scope:**

Technologies used in our solution:

* Webapp:
  + Backend: Flask framework for web app
  + Front-end: Bootstrap for JavaScript, html and CSS used for front-end, along with some custom CSS
* Model:
  + Python-docx : for parsing word documents
  + Pdftotext : for parsing pdf documents
  + Scikit-learn: for various preprocessing, models, and building pipelines
  + Pandas: for data manipulation
  + Keras for experimenting with neural network models
  + Natural Language Toolkit for preprocessing of text

## **Functionality and User Interface:**

* What type of user interface does the solution provide (e.g. web interface, command line interface).
  + Web-based interface for users to access it using computer, phone or tablet
* What input formats does the solution support? (e.g. PDF or MS Word).
  + Supports both **PDF** and **MS Word**
* How does the solution process batches of documents?
  + Currently doesn’t allow for batch processing

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

* Provide a description of the ways in which the technology leverages AI/ML. Please specify general approaches (supervised, unsupervised) and conceptual description of how these apply to the challenge.

One of the principal tasks of machine learning is text classification. The EULA Challenge 2020 focuses on the legal domain and, in particular, on the classification of clauses in legal documents as acceptable or unacceptable EULA language. An **end-user license agreement** (**EULA**, [/ˈjuːlə/](https://en.wikipedia.org/wiki/Help:IPA/English)) is a legal contract entered into between a software developer or vendor and the user of the software, often where the software has been purchased by the user from an intermediary such as a retailer [Wikipedia, accessed 19 August, 2020]. A EULA specifies in detail the rights and restrictions which apply to the use of the software. In many legal settings, governments and institutions manage hundreds of thousands of EULA agreements per year, depending on the size of the institution. Therefore, automatic categorization of document or specific clauses into acceptable or unacceptable clauses significantly enhances the efficiency of document management and decreases the time spent by legal experts analyzing these documents. The main challenge that our solution intends to address, therefore, is the time it takes for a document reviewer to review a proposed EULA agreement and validate it. EULA reviewers are domain experts. It is noted that when EULA reviewers miss unacceptable clauses during document review, the institution to which the reviewer is accountable may assume undue liability and risk. Therefore, in providing an automated solution for EULA clause document review, our team is cognizant that there is a higher risk in having the system falsely accept EULA clauses than in having the system falsely reject EULA clauses. We do keep in mind, though, the trade-off that a false rejection by the system increases review time, while reducing institutional risk.

The proposed *Gaussian Solutions, llc* response to the EULA Challenge, 2020, is a EULA Clause Classification system in which an inference engine assigns one of two labels to each clause in a user-provided document (aka, *query document*). Since the challenge provided pre-labeled ground-truth clauses, our automated EULA clause classification system utilizes a model trained via **supervised learning methods**.

Recently, several sophisticated frameworks have been proposed to address document classification tasks in other domains. However, complex neural networks such as Bidirectional Encoder Representations from Transformers (BERT; Devlin et al., 2018) and similarly complex network architectures require a lot of data to train, are more sensitive to hyperparameter fluctuations and are susceptible to domains that consist of data with dissimilar characteristics. For this reason, as well as through empirical examination, we have determined it is inappropriate to employ an overly complex neural architecture both due to the nature of this particular problem domain and with the limited availability of labeled training data. Rather, we focus on the relative stochastic underlying word structure of the clauses themselves, creating text embedding vectorizations and deploying supervised kernel methods to train the classification model.

Our data cleaning steps included removing stop words, removing digits and punctuation, lemmatizing tokens (using lemma\_wordnet) and parts of speech tags.

To understand the domain and problem space better, we performed extensive data exploration with the provided training dataset. Our exploration included understanding baseline word level tokenization statistics about the positive and negative examples (where positive examples belong to the ‘reject’ class and negative examples belong to the EULA clause ‘accept’ class), word counts and top word comparative data analytics between the positive/negative example sets. The details of our exploration can be found in eula\_eda.ipynb

Our analysis showed that the positive and negative examples contained highly similar word frequencies and therefore word frequency similarity methods (i.e., methods that account for similarity in token word occurrence frequencies) would be inadequate in this instance. We sought methods that exploit more substantive underlying semantic structure and relations among tokens in the clause. For this reason, beyond tokenizing the clauses, we explored a number of model families for learning word vectors including both global matrix factorization methods and local context window methods including: Term Frequency — Inverse Document Frequency (TFIDF), GloVE (Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. [GloVe: Global Vectors for Word Representation](https://nlp.stanford.edu/pubs/glove.pdf)), and Word2Vec (Le and Mikolov, 2014, using the Google implementation). It was important to explore these to efficiently leverage statistical information as well as utilize local word contexts.

To identify the appropriate model to use (model selection *and* parameter selection), we pursued a ***grid-search cross-validation*** approach in which we evaluated the results of different clause vectorization methods and different models (and model parameters) to identify the model that would not be over-trained given the provided labeled dataset. The model chosen (discussed below) was not necessarily the best relative performer, but had performed best compared with model simplicity and explainability (which our team values). The details of the gridsearch can be found in models.py file under code/training folder

In each of our training settings, we used the above-mentioned vectorization approaches and then passed the vectorized data through several supervised learning methods including Decision Trees, Kernel methods (support vector machine with linear and polynomial kernels) and LSTM. Each of our training settings utilized a ***stratified k-fold*** cross validation approach(CV) in which we split the training set into k smaller sets, the model was trained using k-1 of the folds as training data, and the resulting model was validated on the remaining (kth) part of the data. In our analysis k was chosen as 5 that gave a good split of the vectors, and enough cross validation steps. We used stratified approach to make sure each of the cross-validation steps have class balance similar to the whole set. The performance measure we used in the comparative matrix was the average F-score computed in the loop. This approach, while computationally expensive, was used to be able to make the most out of this inverse inference where the number of positive samples we had was very small.

The results of our search for a good model are given in the following table (only the best model for search in each category).

Table 1 Results of model search phase

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **model** | **vectorizer** | **accuracy** | **precision** | **recall** | **f1** | **brier** | **auc** |
| SVM2 | TFIDF-2 | 0.8107 | 0.7127 | 0.7789 | 0.7333 | 0.109 | 0.8694 |
| SVM3 | TFIDF-3 | 0.8147 | 0.7137 | 0.771 | 0.7332 | 0.1095 | 0.8694 |
| SVM1 | TFIDF-1 | 0.8107 | 0.7081 | 0.7632 | 0.7269 | 0.1112 | 0.8604 |
| SGD2 | TFIDF-2 | 0.7518 | 0.6682 | 0.748 | 0.6796 | 0.1801 | 0.8151 |
| SGD3 | TFIDF-3 | 0.7254 | 0.659 | 0.7464 | 0.6616 | 0.2025 | 0.8283 |
| AdaBoost1 | TFIDF-1 | 0.7599 | 0.6701 | 0.7446 | 0.6837 | 0.2423 | 0.8251 |
| AdaBoost2 | TFIDF-2 | 0.7614 | 0.6689 | 0.7403 | 0.683 | 0.2429 | 0.8321 |
| SGD1 | TFIDF-1 | 0.7274 | 0.6556 | 0.7382 | 0.66 | 0.1916 | 0.8204 |
| AdaBoost3 | TFIDF-3 | 0.7756 | 0.6694 | 0.7229 | 0.6847 | 0.2441 | 0.8219 |
| LSTM | Glove | 0.812 | 0.6434 | 0.4049 | 0.496909 | 0.15957 |  |

Details of vectorizers

* TFIDF-1: Term Frequency Inverse Document frequency with max\_features = 500, and uses single word tokens
* TFIDF-2: Term Frequency Inverse Document frequency with max\_features = 2000, and uses monograms as well as bigrams, along with min\_df=3, max\_df=0.98
* TFIDF3: Term Frequency Inverse Document frequency with max\_features = 4000, and uses monograms as well as bigrams, along with min\_df=3, max\_df=0.98

Details of models

* SGD1:

Pipeline(steps=[('standardscaler', StandardScaler(with\_mean=False)),

('sgdclassifier',

SGDClassifier(alpha=0.01, class\_weight='balanced',

loss='modified\_huber', max\_iter=500,

random\_state=3))])

* SGD2:

Pipeline(steps=[('standardscaler', StandardScaler(with\_mean=False)),

('sgdclassifier',

SGDClassifier(alpha=0.1, class\_weight='balanced',

loss='modified\_huber', max\_iter=500,

penalty='elasticnet', random\_state=3))])

* SGD3:

Pipeline(steps=[('standardscaler', StandardScaler(with\_mean=False)),

('sgdclassifier',

SGDClassifier(alpha=1, class\_weight='balanced', loss='log', max\_iter=500, random\_state=3))])

* ADA1

AdaBoostClassifier(base\_estimator=DecisionTreeClassifier(class\_weight=' balanced', max\_depth=1, max\_features=10, random\_state=11),

learning\_rate=0.1, n\_estimators=300, random\_state=3)

* ADA2

AdaBoostClassifier(base\_estimator=DecisionTreeClassifier(class\_weight='balanced', max\_depth=1, max\_features=20, random\_state=11),

learning\_rate=0.1, n\_estimators=300, random\_state=3)

* ADA3

AdaBoostClassifier(base\_estimator=DecisionTreeClassifier(class\_weight='balanced', max\_depth=1, max\_features=10, random\_state=11),

learning\_rate=0.01, n\_estimators=300, random\_state=3)

* SVM1

SVC(C=1, class\_weight='balanced', degree=2, probability=True, random\_state=3)

* SVM2

SVC(C=0.5, class\_weight='balanced', degree=2, probability=True, random\_state=3)

* SVM3

SVC(C=0.5, class\_weight='balanced', degree=2, probability=True, random\_state=3)

* LSTM:
* A recurrent neural network with 2 hidden layers, with 256 and 128 nodes in them, followed by a FCN with 64 nodes, followed by sigmoid operation

After experimenting with various models, we came to the following conclusions:

* SGDx: These models are linear SVM models. As most of the words in both positive and negative classes are similar, it is to be expected that linear models may not perform well.
* ADAx: these models are inherently limited as they make decisions based on rules, and most of the words are same in both the classes, and hence may not have performed as well.
* SVMx: these models are Support Vector Models with non-linear kernels that model the interdependency of the words, and not just one word at a time. These performed the best for the case at hand
* LSTM: RNNs are typically more computationally intensive and perform well given a lot of data. However, in the case at hand, we have limited data, and imbalanced set as well. These models performed the worst and we quickly moved away from them, both because of poor results, and high computational cost

Given the above findings, we chose to use SVM2, along with TFIDF-2 vectorizer for our inference model. This model provided the following scores in our testing.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **accuracy** | **precision** | **recall** | **f1** | **brier** | **auc** |
| 0.8107 | 0.7127 | 0.7789 | 0.7333 | 0.109 | 0.8694 |

Data Augmentation

We used train.pkl file given in the reference file to enhance our training set. This was done by randomly sampling positive clauses, and negative clauses, and using their tokens. However, this approach didn’t result in any improvements to the model accuracy, in fact it reduced the model accuracy and F1 scores. It is possible that the preprocessing steps used in making train.pkl file were quite different from what we use, and hence the intermingling of datasets didn’t result in good results.

We believe more relevant data will improve model. A good way to obtain it is to get legal reviewers contribute to the processing of labeling data. Our web based app incorporates this by providing a feature where reviewers can edit the recommendations from AI engine, and feedback the new labels. These new labeled data will be used to train the AI model, and deploy the new model if improvements to the performance scores are seen. We believe in continuous improvement and continuous deployment model.

Screenshots of the web-based app

Figure 1 Home page of our application

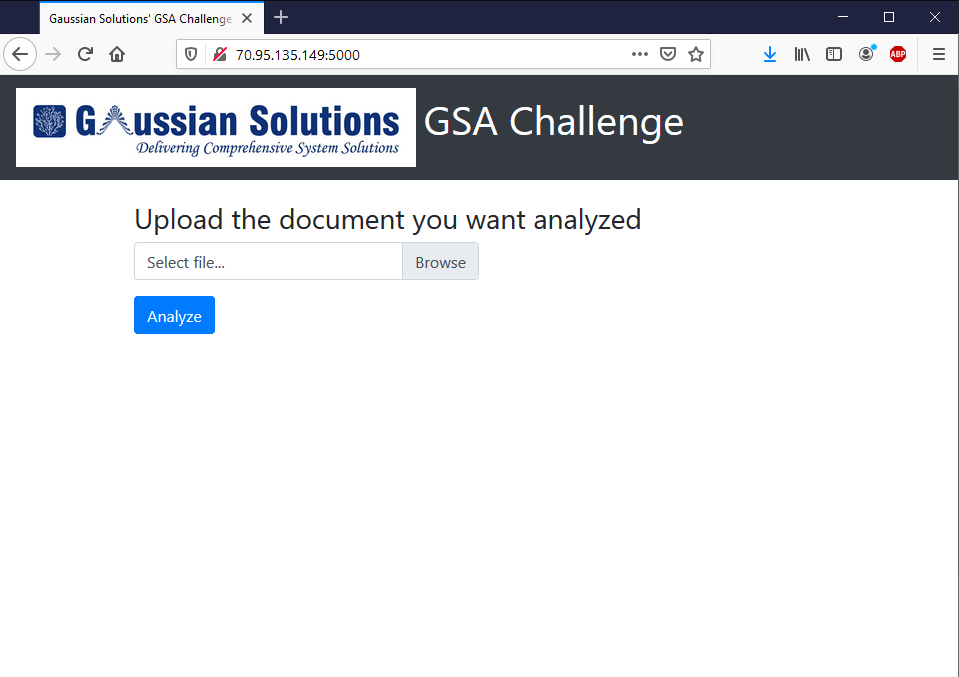


Figure 2 About to upload document for Analysis

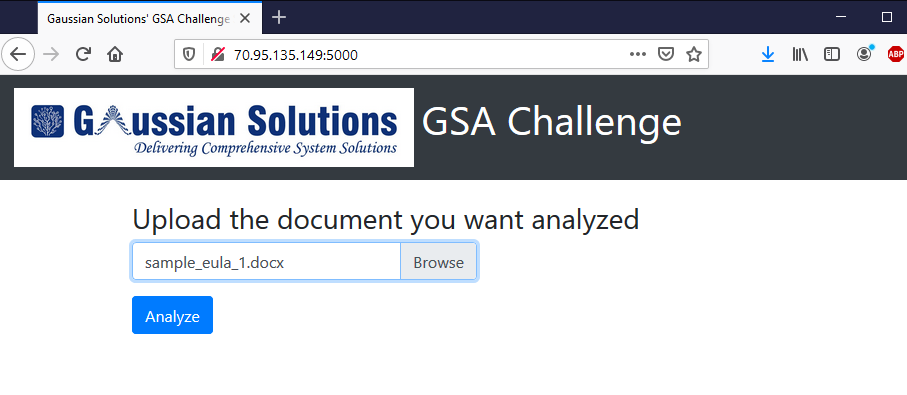
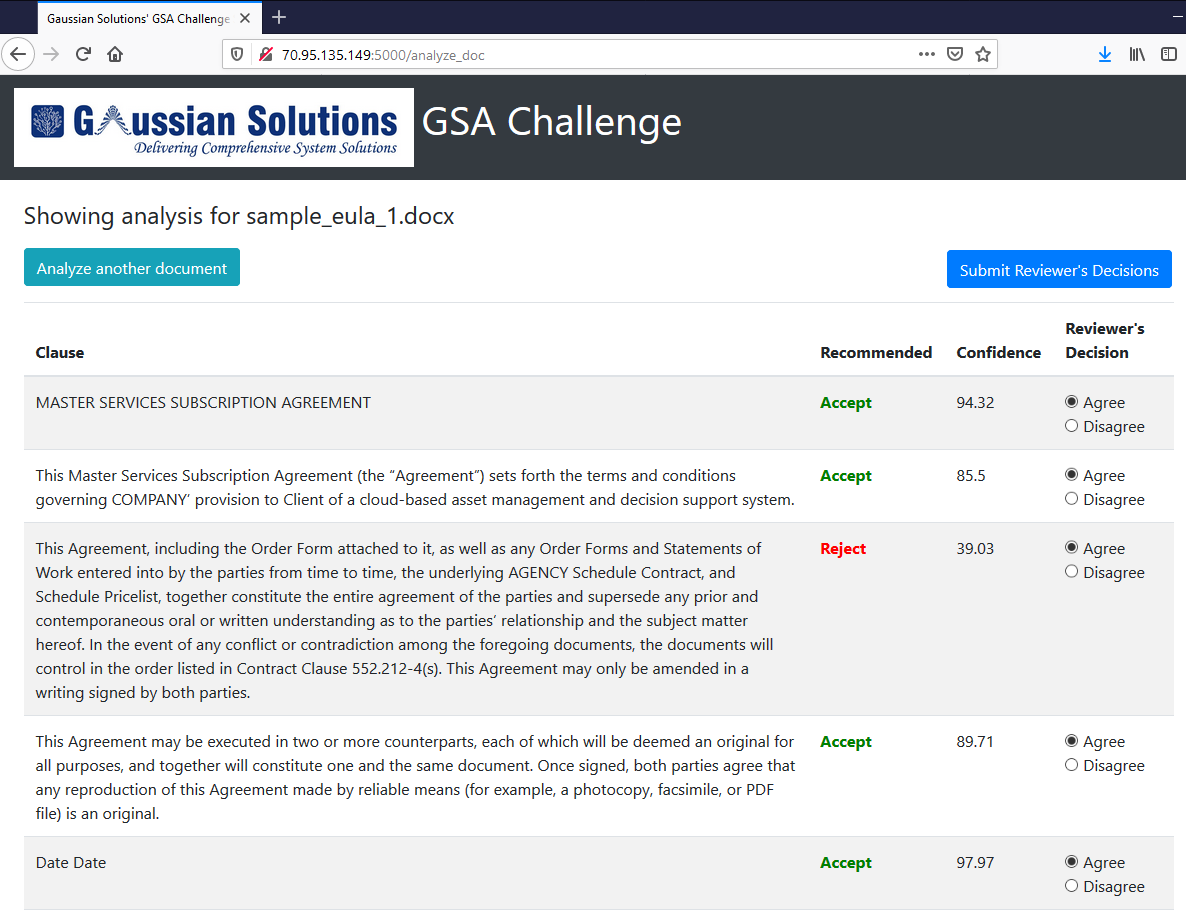


Figure 3 Snapshot of analysis results



A video explaining our solution is uploaded to the github for review.

Future developments

We plan to provide these improvements in future:

* A word add-on plugin to autoextract, and classify the clauses in the word document itself, without having to upload it to the servers
* A command line tools to batch process the documents
* Use law2vec embeddings to see if model performance increases
* Provide the ability to ‘aggressiveness’ of the model, to adjust whether to let more clauses be rejected to avoid unacceptable clauses sneak in as acceptable clauses.

Sincere thanks from our side to let us participate in this challenge.