# GSA EULA Challenge

## Project Overview

EULA stands for end-user license agreements. It details the rights and restrictions which apply to the use of software service. On average it takes all parties involved approximately 7-14 days to review and EULA documents and ensure all unacceptable terms and conditions are identified. This project is to use machine learning algorithms to train a model that can detect acceptable clauses from EULAs files uploaded by users, and also provide confidence score of predictions.

## Data Introduction

#### Training Data : Datasets Provided by GSA

1. training data in csv: AI_ML_Challenge_Training_Data_Set_1_v1.csv (https://github.com/GSA/ai-ml-challenge-2020/blob/master/data/AI_ML_Challenge_Training_Data_Set_1_v1.csv)
2. unacceptable clauses in PDF: appendix_b_unacceptable_clauses in pdf (https://github.com/GSA/ai-ml-challenge-2020/blob/master/reference/appendix_b_unacceptable_clauses.pdf)

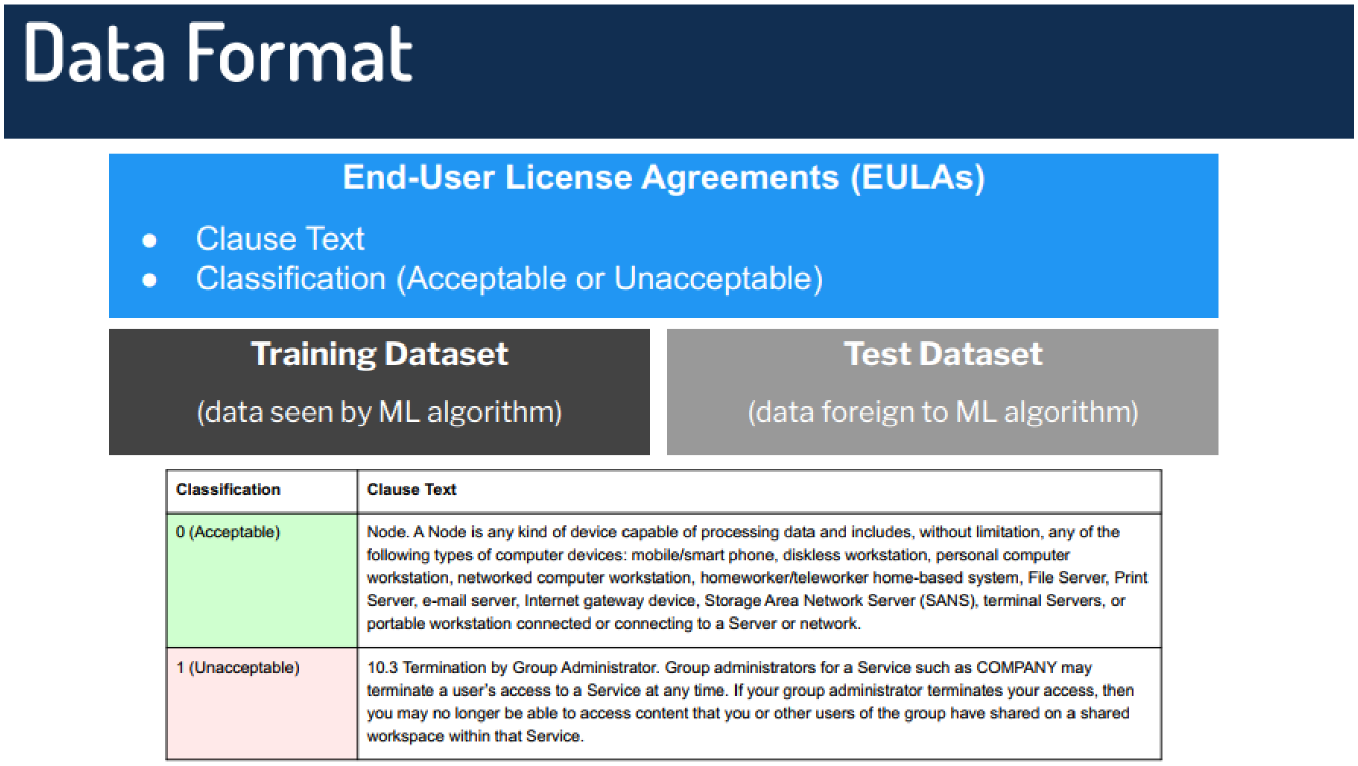
| Clause ID | Clause Text | Classification |
|---|---|---|
| Digits | String/Text data | 0 for Acceptable Clause; 1 for Unacceptable Clause |

In [2]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import pandas as pd
# TODO: setting for training model
upload_data = 'attachments/'
from IPython.display import Image
training_data = pd.read_csv(upload_data+'AI_ML_Challenge_Training_Data_Set_1_v1.txt',sep
=',')
appendix= pd.read_excel(upload_data+'Clauses_From_Appendix.xlsx')
training_data = pd.concat([training_data,appendix],axis=0).reset_index(drop=True)
training_data['Clause ID'] = training_data['Clause ID'].fillna(training_data[training_da
ta['Clause ID'].isna()]['Clause ID'].index.to_series())
print('Training data size: ',training_data.shape)

Image(filename=upload_data+'data_format.png')
```

```
Training data size:  (7893, 3)
```

Out[2]:



Data Format

End-User License Agreements (EULAs)

- Clause Text
- Classification (Acceptable or Unacceptable)

Training Dataset

(data seen by ML algorithm)

Test Dataset

(data foreign to ML algorithm)

| Classification | Clause Text |
|---|---|
| 0 (Acceptable) | Node. A Node is any kind of device capable of processing data and includes, without limitation, any of the following types of computer devices: mobile/smart phone, diskless workstation, personal computer workstation, networked computer workstation, homeworker/teleworker home-based system, File Server, Print Server, e-mail server, Internet gateway device, Storage Area Network Server (SANS), terminal Servers, or portable workstation connected or connecting to a Server or network. |
| 1 (Unacceptable) | 10.3 Termination by Group Administrator. Group administrators for a Service such as COMPANY may terminate a user's access to a Service at any time. If your group administrator terminates your access, then you may no longer be able to access content that you or other users of the group have shared on a shared workspace within that Service. |

## Data Processing

### PDF/Word Text Extraction

The application accepts EULAs in PDF or MS word fomart. It can parse multiple documents at one-time upload and get predictions.

This step is to extract clause from pdf or word files provided by GSA. Files would be used as testing data to get predictions from the model. After parsing file, processed text would been identified as clauses or subtitles. Only clauses would be predicted as accepted/unaccepted.

### Split Clauses and Subtitles

After parsing documents into rows of string which are clauses or subtitles, each line in the row will be checked if it is a signature area based on the existence of trailing colon and words that begin with letter in the uppercase. If yes, it will be removed from the row.

**Used packages**:

> PDF EULA: `PyMuPDF` package is used to read the PDF into raw string. Item numbers such as “1.1”, “a.” and “i.” are used in the re package to split the string of PDF documents into clauses and subtitles.
>
> MS Word EULA: `docx` is used to read MS Word documents. Files are converted into strings delimited by paragraph, which is used to split the string into clauses and subtitles.

*Source Code: python files under* `/src/`

**NLP Process**

- Remove irrelevant patterns, special characters, punctuations, stopwords

```
    irrelevant patterns:  Item number such as “2.7” inside the clauses
```

- Text Lemmatization
- TF-IDF Vectorizer

*Source Code:* *`modules/nlp.py`*

**Text Augmentation**

Due to lack of large data for unacceptable clauses to be learnt, we use text augmentation to generate more text data by finding synonyms from Wordnet in NLTK, which is a popular open-source lexical database for the English. It groups words like nouns, verbs, adjectives and adverbs into sets of cognitive synonyms called synsets each expressing different concept, provides short definitions and usage examples, and records a number of relations among these synonym sets.

Steps:

```
Import NLTK's wordnet;
Find 3 words from wordnet for each word in the unaccpetable clauses and generate new sentence
s;
Remove duplicated clauses and merge these new clauses with original training data.
```

Augmented Results:

```
Generated 3,482 unacceptable clause data
```

*Source Code:* *`modules/text_augmentation.py`*

## Model

### Model Methods and Steps

```
i. Pretrain Models and Decide the Best Model
    1. Split cleaned labelled-data into 85% training data and 15% testing data(unseen data)
    2. Apply text augmentation on the training data
    3. Perform TfidfVectorizer on training data
    4. Use Synthetic Minority Over-sampling Technique(SMOTE) to increase minority class, whic
h is 1(unacceptance)
    5. Fit into models and use GridSearchCV to get best parameters
    6. Try different combinations of resample size (0.3, 0.5, 0.7, 1) and number of synonyms
 (0-5) in text augmentation
    7. Evaluate models by Brier Scores and F1 Score, and decide the best model
ii. Use the Best Model to Predict Validation Data
    8. Fit the best model with all training data(training + testing data) to predict the vali
dation data provided by GSA
    9. Get both classification results and the probability of predictions
```

### Model Performane and Selection

Model sets: XGBoost, Logistic Regression, KNN, MLP, Random Forest, Naive Bayes

Best model: `Random Forest`

Evaluation Metrics:

```
F1 Score :
```

$$F1\ Score = 2 * \frac{(recall * precision)}{(recall + precision)}$$

```
Brier Score
```

$$Brier\ Score = \frac{1}{n} * \sum_{i=0}^{n} (ft - ot)^2$$

```
ft = predicted probability of 1

ot = prediction (0 or 1)

n = number of instances
```

- For i. Model Selection, Source Code: `pretrained_improve.py` *\ Please note: parameters tuning, resample size or augmentation synonyms setting needs to change code manually.
- For ii.Best Model, Source Code: `main.py, pretrained.py` *

#### Self-reported Metric

Random Forest outperforms amongst all six models. And its performance is:

- F1 Score:0.54
- Brier Score:0.11

*Source Code: `pretrained_improve.py`*

In [26]:
```
pd.set_option('max_colwidth',0)
srm = pd.read_csv(upload_data+'Self_Reported_Metrics.csv')
srm
```

Out[26]:

| | Unnamed: 0 | brier_score | f_score | recall | precision | accuracy | true_positive | true_negative | false_positiv |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Random Forest | 0.110242 | 0.543307 | 0.492857 | 0.605263 | 0.853165 | 69 | 605 | 4 |

#### Validation Prediction and Model Inference

For more detailed explanation on predictions of validataion, **please refer to the Description of Methods 02.pdf**(./Description of Methods 02.pdf)

---

## Database

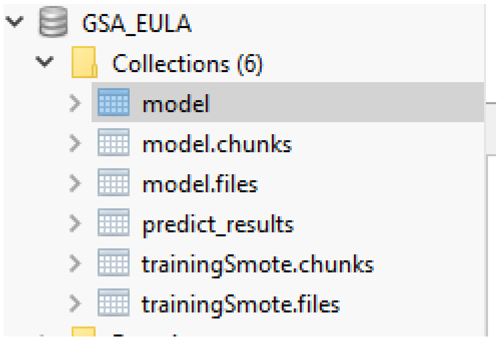
#### MongoDB

The application uses MongoDB as the database to save intermediate results, including:

1. model - model name, best parameters, vectorizer
2. model.files + model.chunks - model in GridFS format (in order to save the actual model object and call it)
3. predict_results - clause text, prediction, probability of prediction
4. trainingSmote.files + trainingSmote.chunks - training data in GridFS format after resampling

In [9]:
```python
Image(filename=upload_data+'db1.png')
#database catalog
```

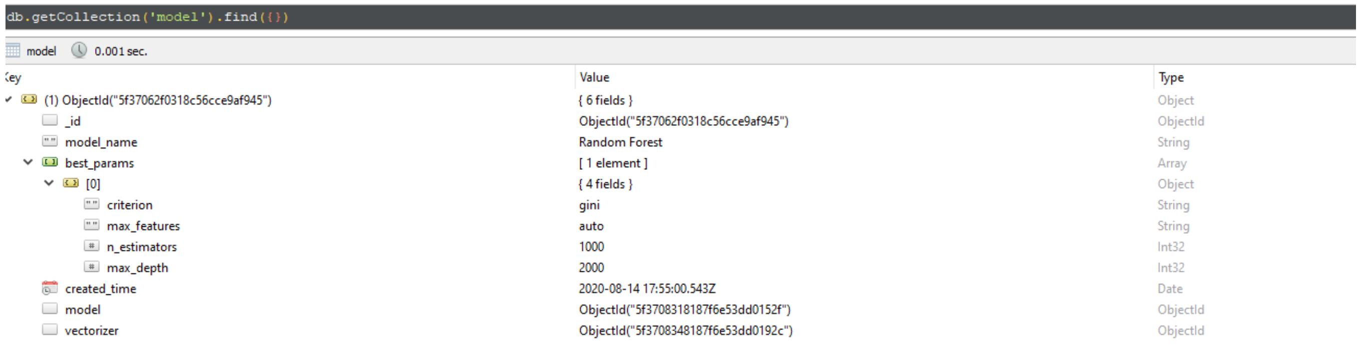
Out[9]:



```
GSA_EULA
  Collections (6)
    model
    model.chunks
    model.files
    predict_results
    trainingSmote.chunks
    trainingSmote.files
```

In [8]:
```python
Image(filename=upload_data+'db2.png')
#model
```

Out[8]:



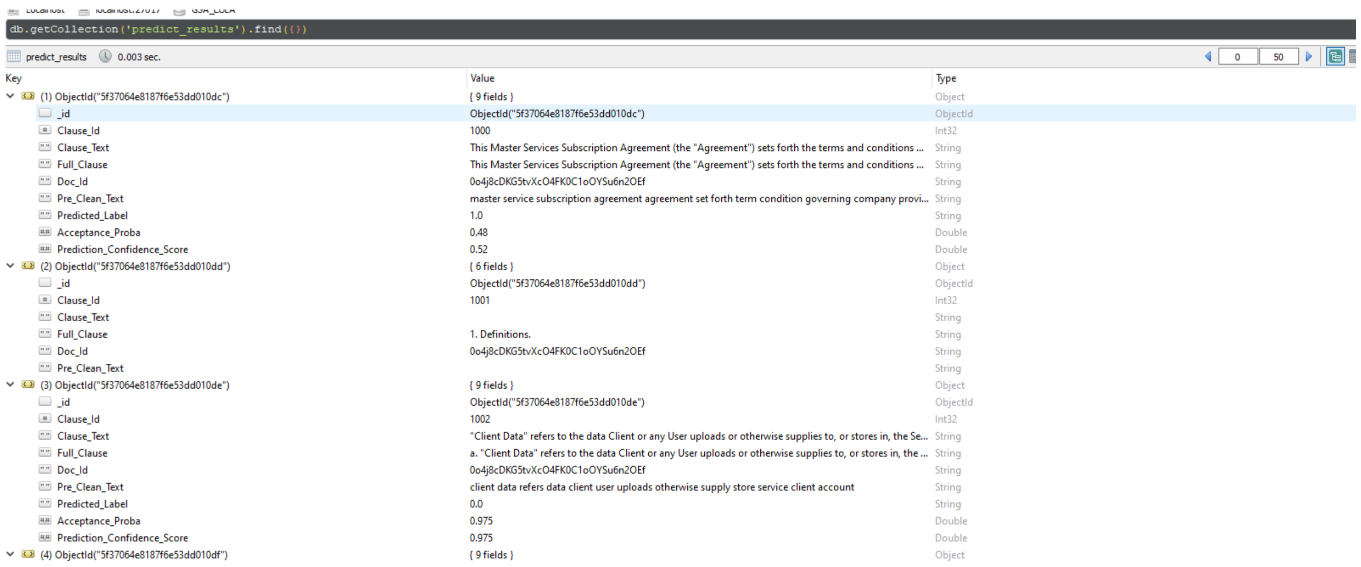
```
db.getCollection('model').find({})

model   0.001 sec.

Key                                          Value                                      Type
(1) ObjectId("5f37062f0318c56cce9af945")     { 6 fields }                               Object
    _id                                      ObjectId("5f37062f0318c56cce9af945")       ObjectId
    model_name                               Random Forest                              String
    best_params                              [ 1 element ]                              Array
        [0]                                  { 4 fields }                               Object
            criterion                        gini                                       String
            max_features                     auto                                       String
            n_estimators                     1000                                       Int32
            max_depth                        2000                                       Int32
    created_time                             2020-08-14 17:55:00.543Z                   Date
    model                                    ObjectId("5f3708318187f6e53dd0152f")       ObjectId
    vectorizer                               ObjectId("5f3708348187f6e53dd0192c")       ObjectId
```

In [10]:
```python
Image(filename=upload_data+'db3.png')
#predict_results
```

Out[10]:



```
db.getCollection('predict_results').find({})

predict_results   0.003 sec.                                                               0   50

Key                                          Value                                                                                        Type
(1) ObjectId("5f37064e8187f6e53dd010dc")     { 9 fields }                                                                                 Object
    _id                                      ObjectId("5f37064e8187f6e53dd010dc")                                                         ObjectId
    Clause_Id                                1000                                                                                         Int32
    Clause_Text                              This Master Services Subscription Agreement (the "Agreement") sets forth the terms and conditions ...   String
    Full_Clause                              This Master Services Subscription Agreement (the "Agreement") sets forth the terms and conditions ...   String
    Doc_Id                                   0o4j8cDKG5tvXcO4FK0C1oOYSu6n2OEf                                                             String
    Pre_Clean_Text                           master service subscription agreement agreement set forth term condition governing company provi...   String
    Predicted_Label                          1.0                                                                                          String
    Acceptance_Proba                         0.48                                                                                         Double
    Prediction_Confidence_Score              0.52                                                                                         Double
(2) ObjectId("5f37064e8187f6e53dd010dd")     { 6 fields }                                                                                 Object
    _id                                      ObjectId("5f37064e8187f6e53dd010dd")                                                         ObjectId
    Clause_Id                                1001                                                                                         Int32
    Clause_Text                                                                                                                           String
    Full_Clause                              1. Definitions.                                                                              String
    Doc_Id                                   0o4j8cDKG5tvXcO4FK0C1oOYSu6n2OEf                                                             String
    Pre_Clean_Text                                                                                                                        String
(3) ObjectId("5f37064e8187f6e53dd010de")     { 9 fields }                                                                                 Object
    _id                                      ObjectId("5f37064e8187f6e53dd010de")                                                         ObjectId
    Clause_Id                                1002                                                                                         Int32
    Clause_Text                              "Client Data" refers to the data Client or any User uploads or otherwise supplies to, or stores in, the Se...   String
    Full_Clause                              a. "Client Data" refers to the data Client or any User uploads or otherwise supplies to, or stores in, the ...   String
    Doc_Id                                   0o4j8cDKG5tvXcO4FK0C1oOYSu6n2OEf                                                             String
    Pre_Clean_Text                           client data refers data client user uploads otherwise supply store service client account   String
    Predicted_Label                          0.0                                                                                          String
    Acceptance_Proba                         0.975                                                                                        Double
    Prediction_Confidence_Score              0.975                                                                                        Double
(4) ObjectId("5f37064e8187f6e53dd010df")     { 9 fields }                                                                                 Object
```

---

## Software and Coding Language

1. Python is used for the backend design which includes the parsing of documents, natural language processing on clauses, text augmentation, training machine learning models and creating RESTful APIs in FLASK.
2. Agular JS is used for the front-end web application building.
3. MongoDB is used to store all the original parsed clauses, cleaned clauses, predicted results and machine learning model.
4. Postman is used to simulate the interaction between frontend and backend for the test purpose.
5. AWS (Amazon Web Service） is used for deployment of this application includes front-end, back-end and database.