greatlearning

CAPSTONE PROJECT - NLP CHATBOT

INTERIM REPORT

14th February 2021

By:

Kajal Jaiswal

Damini Tiwari

Ashwini Kumar

Anis Uddin

TABLE OF CONTENTS

Summary of Problem Statement, Data and Findings

Health and Safety is of the utmost importance in the industrial field especially when workers are interacting with heavy machinery and equipment on a daily basis. Even though most industries have stringent policies as well as rules and regulations in place to ensure worker safety, accidents can happen at any time and place. Receiving immediate support after the occurrence of an accident is essential and can save lives if the accurate information is available on hand and the correct procedures are implemented. As time is of the essence in such critical instances, a smart chatbot that can be pinged for such support would add great value and help in recognizing the risks involved as well as the correct procedures to be followed thereafter.

Artificial intelligence has made chatbots more lifelike than ever before. A chatbot is a computer program that imitates human conversation — spoken, written, or both. Chatbots conduct conversations with people, and in ideal situations, users would not need to realize that they're actually talking to a bot. AI, using tools and modules such as NLP, ML, DNN make it possible for chatbots to "learn" by discovering patterns in the given textual data. With the appropriate training, these chatbots can then apply the patterned learning to similar situations and provide responses which are relevant to the context right away. This ability gives them the "intelligence" to perform tasks, recognize & solve problems, and manage information without human intervention.[1]

Our AIML Capstone project is to design a ML/DL based chatbot utility which can help the professionals to highlight the safety risk as per the incident description. The database comes from one of the biggest industries in Brazil and in the world. It is an urgent need for industries/companies around the globe to understand why employees still suffer some injuries/accidents in plants.

Goals

1. Designing an AIML chatbot that recognizes the Accident Level, Potential Accident Level, Critical Risks involved with the accident based on the descriptions provided.
2. Designing and deploying a clickable UI based chatbot interface.

Data Specifications

The database is the records of accidents from 12 different plants in 3 different countries. The data is of the shape: 425 rows and 11 columns. Every row in the dataset is an occurrence of an accident along with 11 features which are described further below.

Columns description:

- **Data**: timestamp or time/date information
- **Countries**: which country the accident occurred (anonymised)
- **Local**: the city where the manufacturing plant is located (anonymised)
- **Industry sector**: which sector the plant belongs to
- **Accident level**: from I to V, it registers how severe was the accident (I = not severe, V = very severe)

- **Potential Accident Level**: Depending on the Accident Level, the database also registers how severe the accident could have been (ranging from I to VI, where I = not severe, VI = very severe))
- **Genre**: if the person is Male or Female
- **Employee or Third Party**: if the injured person is an Employee or a Third Party
- **Critical Risk**: some description of the risk involved in the accident
- **Description**: Detailed description of how the accident happened

Milestones

1. Data preparation (cleansing & preprocessing) to be used for AIML model learning.
2. Design, train and test multiple ML, NN and LSTM classifiers and choose the best performing model classifier.
3. Clickable UI based chatbot interface which accepts text as input and replies back with relevant answers.

Summary of the Approach to EDA and Preprocessing

	Data	Countries	Local	Industry Sector	Accident Level	Potential Accide	Genre	Employee or Thir	Critical Risk	Description
0	2016-01-01 0:00:	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 for maintenance, the supe
1	2016-01-02 0:00:	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Sys	During the activation of a sodium sulphide pump, the piping was uncou
2	2016-01-06 0:00:	Country_01	Local_03	Mining	I	III	Male	Third Party (Rem	Manual Tools	In the sub-station MILPO located at level +170 when the collaborator w
3	2016-01-08 0:00:	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 CX-695 OB7, the person
4	2016-01-10 0:00:	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances that the mechanics Ant
5	2016-01-12 0:00:	Country_02	Local_05	Metals	I	III	Male	Third Party (Rem	Pressurized Sys	During the unloading operation of the ustulado Bag there was a need to
6	2016-01-16 0:00:	Country_02	Local_05	Metals	I	III	Male	Employee	Fall prevention (s	The collaborator reports that he was on street 09 holding in his left han
7	2016-01-17 0:00:	Country_01	Local_04	Mining	I	III	Male	Third Party	Pressed	At approximately 04:50 p.m., when the mechanic technician José of th
8	2016-01-19 0:00:	Country_02	Local_02	Mining	I	IV	Male	Third Party (Rem	Others	Employee was sitting in the resting area at level 326 (raise bore), wher
9	2016-01-26 0:00:	Country_01	Local_06	Metals	I	II	Male	Third Party	Chemical substa	At the moment the forklift operator went to manipulate big bag of bioxic
10	2016-01-28 0:00:	Country_01	Local_03	Mining	I	III	Male	Employee	Others	While installing a segment of the polyurethane pulley protective lyner -

Fig 1: Glimpse of raw data

Preliminary Exploration

There are 11 columns in the given dataset including the unnamed serial column. Out of these, the relevant columns that will be used for our model are 'Accident Level', 'Potential Accident Level', 'Critical Risk' and 'Description'.

The other columns, however, can be used for certain insights such as the correlations of highest accidents with the industry sector as well as visualizing any other insights that can be extracted.

We checked for any null values or missing data and found that there are none as shown below.

Table 1: Null value counts

Column	Null values
Unnamed	0
Data	0
Countries	0
Local	0
Industry Sector	0
Accident Level	0
Potential Accident Level	0
Genre	0
Employee or Third Party	0
Critical Risk	0
Description	0

Now, let's look at the unique classes of the columns available.

Table 2: Unique classes

Column	Unique Class
Industry Sector	Mining', 'Metals', 'Others'
Accident Level	'I', 'IV', 'III', 'II', 'V'
Potential Accident Level	'IV', 'III', 'I', 'II', 'V', 'VI'
Employee or Third Party	'Third Party', 'Employee', 'Third Party (Remote)'
Critical Risk	'Pressed', 'Pressurized Systems', 'Manual Tools', 'Others', 'Fall prevention (same level)', 'Chemical substances', 'Liquid Metal', 'Electrical installation', 'Confined space', 'Pressurized Systems / Chemical Substances', 'Blocking and isolation of energies', 'Suspended Loads', 'Poll', 'Cut', 'Fall', 'Bees', 'Fall prevention', '\nNot applicable', 'Traffic', 'Projection', 'Venomous Animals', 'Plates', 'Projection/Burning', 'remains of choco', 'Vehicles and Mobile Equipment', 'Projection/Choco', 'Machine Protection', 'Power lock', 'Burn', 'Projection/Manual Tools', 'Individual protection equipment', 'Electrical Shock', 'Projection of fragments'

Visualizations & Insights

Exploring data requires visualizations to make inferences that help with the model building process. Below we look dive into the significant visuals that describe the data along with the insights gathered.

'Accident Level' with respect to 'Accident Level' count

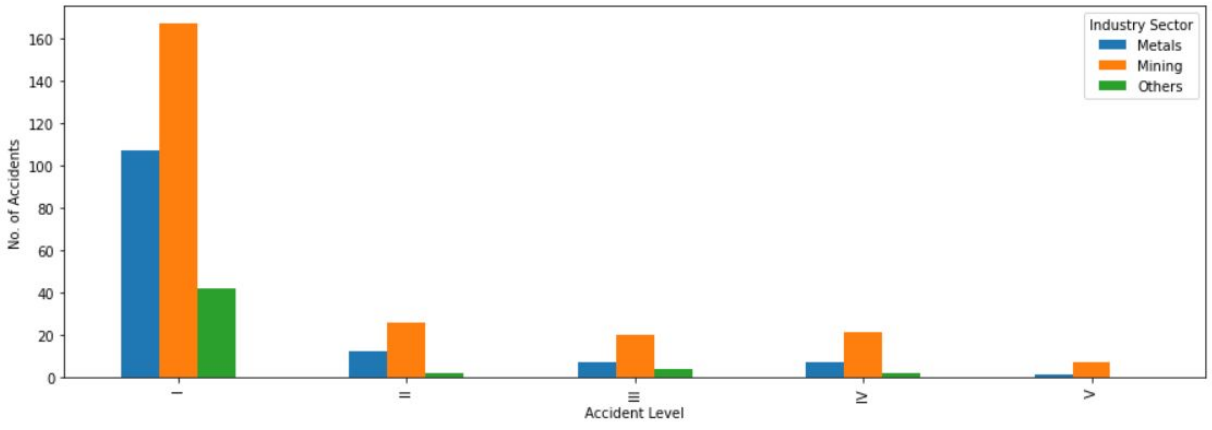


Fig 2: 'Accident Level' vs Accident Level Counts

We can see that there is a significantly larger frequency of Accident Level = I and Accident Levels II, III and IV are very similar in frequency. Accident Level = V is the lowest count.

We can infer that as there are multiple stringent rules and regulations already in place, the highly severe accidents are curbed and it is mostly the non-severe accidents that are captured. We can also make an assumption that a bias exists in categorizing the Accident Levels and it is more likely that an Accident Level of severity II or even III might be reported and recorded as Accident Level I. This can be due to lack of clear distinctions in classifying the accidents or even an internal motivation to showcase very low numbers of Accident Level II and above. However, we have to take the data as it is and to perform intrusive processing in order to balance the data could lead in tampering the integrity of the outcome as well.

We can also look into segregation based on 'Industry Sector' in the above visualization. This clearly shows us that Mining is the most dangerous Industry Sector with the highest frequency of accidents reported across any of the 'Accident Levels'. The second most dangerous 'Industry Sector' would be Metals and lastly Others is the lowest reported accident counts.

'Potential Accident Level' with respect to 'Potential Accident Level' counts.

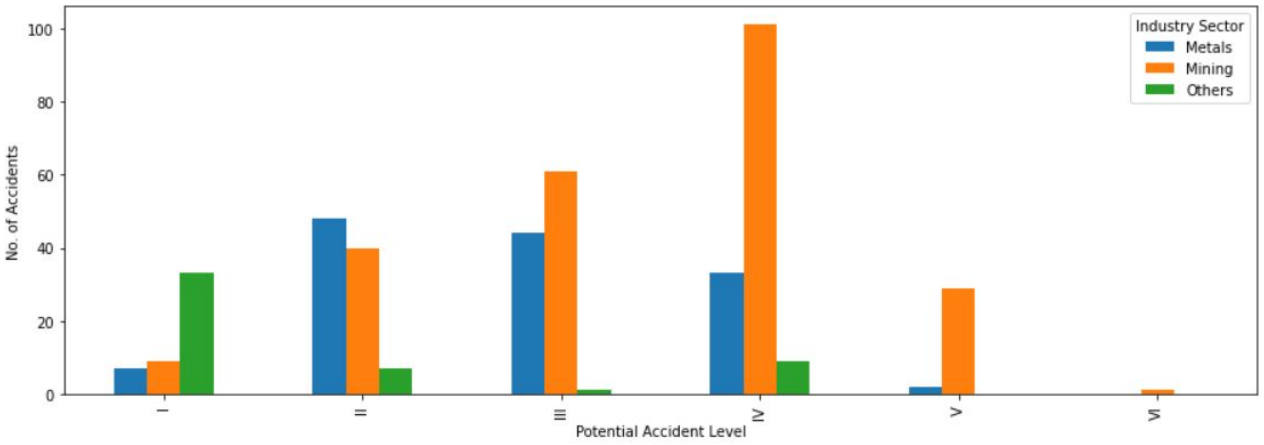


Fig 3: 'Potential Accident Level' vs 'Potential Accident Level' Counts

The visualization of 'Potential Accident Level' against the frequency of the same gives us a completely different picture than the 'Accident Level' discussion above. Here we see how severe the accidents *could* have been. This goes to describe that the safety rules and regulations put in place are actually helping in reducing the severity of the accidents. We can see that IV is the most frequent 'Potential Accident Level' for the Mining Industry Sector which confirms again that Mining is the most dangerous Industry Sector with not only the highest frequency of reported accidents but it is also potentially the most severe level of accidents that can take place - it is worth noting that only Mining Industry Sector is recorded in Level VI. As for the Metals Industry Sector, the 'Potential Accident Level' frequency peaks at Level II and gradually decreases, which goes to show that enforcing safety rules and regulations in this Industry Sector is more effective in curbing increasingly severe accidents. Lastly, we can see that the Other Industry Sector peaks at Level I and significantly diminishes over Levels II and III and rises again over level IV. This shows that in the Other Industry Sector, accidents are more likely to be either mostly non-severe or significantly severe with only accidents which are far and few between Levels I and IV.

As a metric, we can assume that accidents are more likely to be labelled as potentially more severe as there is always something worse that *could* have happened. However, we must avoid tampering with this spread of data as well since it is based on the input from industry experts who can from their experience predict the potential severity of an accident.

Count Plots for 'Accident Level' and 'Potential Accident Level'

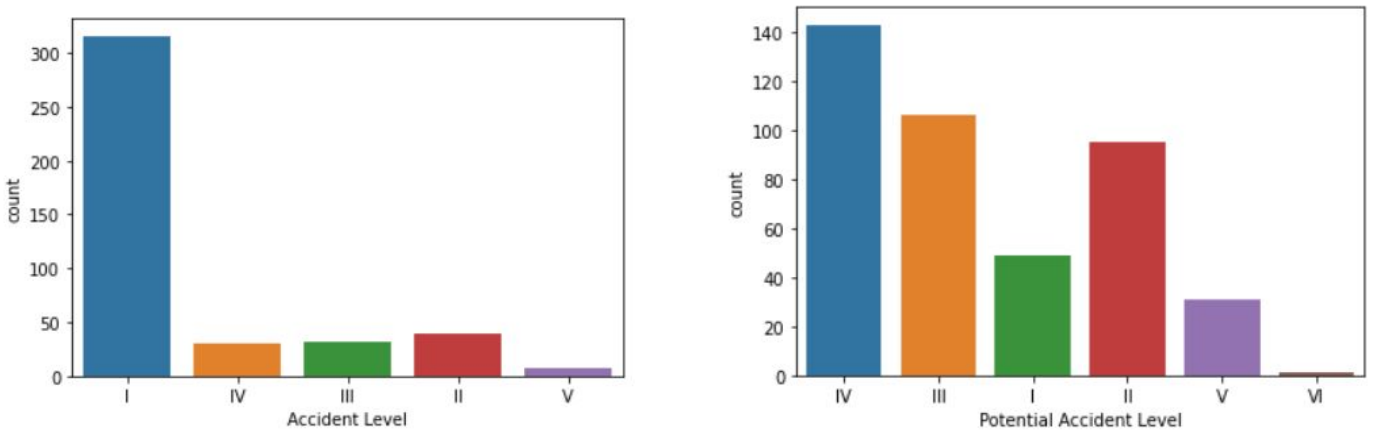


Fig 4: Count Plots for 'Accident Level' and 'Potential Accident Level'

Looking at both 'Accident Level' and 'Potential Accident Level' frequency plots side by side gives us the picture that both reported accident counts and potential accident counts goes down as severity increases. It is more significantly depicted in the 'Accident Level' feature as discussed previously.

EDA and preprocessing related to 'Description' feature

The key column that requires preprocessing is the 'Descriptions' column that is our independent variable. As this is an all texts column, we would be applying NLP preprocessing to prepare this for use in our models.

Firstly, let us visualize the frequency of the text within this column by using a Word Cloud. We have removed the stop words and punctuations for this representation. The Word Cloud is depicted below.

```
# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

Fig 5: Code for Plotting Word Cloud

Fig 6: Word Cloud for 'Description'

The words that stand out right away are: employee, collaborator, operator, moment, activity, assistant, worker, equipment, accident, causing and floor. This gives us an idea of the kind of language that is being used in the description of the accidents and as expected, the highest frequency words are objects and subjects that describe the 'who' part of the description. The secondary words can be noted as the verbs and nouns such as area, causing, foor, injury, using, performing, cleaning, maintenance, hand, platform etc which describe the actions and locations of where these accidents might have taken place.

As a visual, the Word Cloud gives us a comprehensive glimpse into the structure of text in 'Descriptions' feature and gives the assurance that the majority of the matter is captured in these words. Hence we can start to perform our NLP preprocessing to prepare the text data for our models.

Preprocessing of 'Descriptions' for NLP application

Firstly, we will eliminate any special characters such as @,#,$,%,^,&,*,(,) and other such special characters such as in the words: Fernández, José, Aripuanã for example. Also, we will eliminate numbers from the descriptions as they do not carry any value in textual descriptions where you would need to comprehend the units as well as the contexts of those numbers.

In other words, we will keep characters A - Z and a - z only.

We will also remove Stopwords. Stopwords are the most common words in any natural language. For the purpose of analyzing text data and building NLP models, these stopwords might not add much value to the meaning of the descriptions. We will be using NLTK's Stopwords for English. Below are the included words in this array :

["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during", "before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under", "again", "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each", "few", "more", "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s", "t", "can", "will", "just", "don", "should", "now"] [2]

Lastly, we will also remove all punctuation marks such as . , , ! , :, ;, “, ‘

```
import re

# will take only alphabets + will remove any unwanted special characters
data.Description = data.Description.apply(lambda x: re.sub('[^A-Za-z]+',' ',x))

data.Description = data.Description.apply(lambda x: x.lower())

data.Description = data.Description.apply(lambda x : x.strip())

#Removing StopWords
import nltk
import string
nltk.download('stopwords')
from nltk.corpus import stopwords

stop_words = stopwords.words('english') + list(string.punctuation)

data.Description = data.Description.apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))

#lemmatization
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
data.Description = data.Description.apply(lambda x: ' '.join([lemmatizer.lemmatize(w) for w in x.split()]))
```

Fig 7: NLP Preprocessing Code Block

Lemmatization

In the above block of code, we have also implemented lemmatization. Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Basically, it is the process of converting a word to its base form. Moreover, Wordnet is a large, freely and publicly available lexical database for the English language aiming to establish structured semantic relationships between words. It offers lemmatization capabilities as well and is one of the earliest and most commonly used lemmatizers [3].

```
#lemmatization
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
data.Description = data.Description.apply(lambda x: ' '.join([lemmatizer.lemmatize(w) for w in x.split()]))
```

Fig 8: Lemmatization Code Block

Next, we will apply a spell checker to ensure that erroneous spellings do not impact our model's predictive ability.

```
from spellchecker import SpellChecker
spell = SpellChecker()

def spell_check(x):
    correct_word = []
    mispelled_word = x.split()
    for word in mispelled_word:
        correct_word.append(spell.correction(word))
    return ' '.join(correct_word)
```

Fig 9: Spellchecker Code Block

Also, to decrease the load on our models, we will be eliminating the 20 most rare words from the corpus.

```
freq = pd.Series(' '.join(data['Description']).split()).value_counts()[-20:] # 20 rare words
freq
```

```
freq = list(freq.index)
data['Description'] = data['Description'].apply(lambda x: " ".join(x for x in x.split() if x not in freq))
data['Description'].head()
```

Fig 10: Code for extracting 20 most Rare words

Below is the tabulated result of running the above code for extracting rare words.

Table 3: 20 most Rare words with frequency

Sr	Word	Frequency	Sr	Word	Frequency
1	bowl	1	11	facilitate	1
2	looked	1	12	anthony	1
3	tree	1	13	nasciment	1
4	bricklayer	1	14	absorbing	1
5	slaughter	1	15	blaster	1
6	pear	1	16	clogged	1
7	mona	1	17	attaching	1
8	lodged	1	18	scrap	1
9	locate	1	19	brushcutters	1
10	share	1	20	kept	1

Tokenization

Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens. Performing tokenization allows us to convert the raw text data into applicable tokens that can be analyzed by AIML models. We will use the split() function to tokenize our 'Description' feature. We will define the following values before implementing the tokenizer.

max_features = 10000

maxlen = 50

embedding_size = 200

```
#will only take most frequent words
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(data.Description.values)
X = tokenizer.texts_to_sequences(data.Description.values)
```

Fig 11: Code for Tokenizer application

Sequence Padding

As a last step of NLP preprocessing, we will pad the sentences in the 'Description' feature. The pad_sequences() function in the Keras deep learning library can be used to pad variable length sequences. pad_sequences is used to ensure that all sequences in a list have the same length. By default this is done by

padding 0 in the beginning of each sequence until each sequence has the same length as the longest sequence. We have set our maxlen = 50 which will truncate any sentence which is longer than this. We do this to ensure that our models receive a standardized input to work on. Below is the code showing our padding implementation.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
X = pad_sequences(X,maxlen=50,padding='post')
print(X.shape)
```

Fig 12: Code for sequence padding application

Here we have also defined our 'X ' variable which will be used for data splitting before model building.

We can now add our GloVE embeddings as shown below.

```
import numpy as np
EMBEDDING_FILE = './glove.6B.200d.txt'

embedding_matrix = np.zeros((num_words,200))

embeddings = {}
for s in open(EMBEDDING_FILE):
  word = s.split(" ")[0]
  emd = s.split(" ")[1:]
  emd = np.asarray(emd,dtype='float32')
  embeddings[word] = emd

for word,i in tokenizer.word_index.items():
  embedding_vector = embeddings.get(word)
  if embedding_vector is not None:
    embedding_matrix[i] = embedding_vector
```

Fig 13: Code for GloVE Embeddings

Lastly, we will compare some instances from the raw data to our now preprocessed data to get a visual understanding of the changes made.

Table 4: Comparison of 'Description' before and after Preprocessing

Before Preprocessing	After Preprocessing
data.Description[1] During the activation of a sodium sulphide pump, the piping was uncoupled and the sulfide solution was designed in the area to reach the maid. Immediately she made use of the emergency shower and was directed to the ambulatory doctor and later to the hospital. Note: of sulphide solution = 48 grams / liter.	data.Description[1] activation sodium sulphide pump piping uncoupled sulfide solution designed area reach maid immediately made use emergency shower directed ambulatory doctor later hospital note sulphide solution gram liter

data.Description[2] In the sub-station MILPO located at level +170 when the collaborator was doing the excavation work with a pick (hand tool), hitting a rock with the flat part of the beak, it bounces off hitting the steel tip of the safety shoe and then the metatarsal area of the left foot of the collaborator causing the injury.	data.Description[2] sub station milo located level collaborator excavation work pick hand tool hitting rock part beak bounce hitting steel tip safety shoe metatarsal area left foot collaborator causing injury
data.Description[4] Approximately at 11:45 a.m. in circumstances that the mechanics Anthony (group leader), Eduardo and Eric Fernández-injured-the three of the Company IMPROMEC, performed the removal of the pulley of the motor of the pump 3015 in the ZAF of Marcy. 27 cm / Length: 33 cm / Weight: 70 kg), as it was locked proceed to heating the pulley to loosen it, it comes out and falls from a distance of 1.06 meters high and hits the instep of the right foot of the worker, causing the injury described.	data.Description[4] approximately circumstance mechanic anthony group leader eduardo eric fern nder injured three company improve performed removal pulley motor pump zap marcy am length am weight keg locked proceed heating pulley loosen come fall distance meter high hit instep right foot worker causing injury described
data.Description[12]The collaborator reports that he was working in the Ustulación and realized that the cyclone duct was obstructed and opened the door to try to unclog the same and the material detached and projected towards the employee causing small burn in the right heel.	data.Description[12] collaborator report working ustulaci i realized cyclone duct obstructed opened door try unclog material detached projected towards employee causing small burn right heel

We have now cleansed and preprocessed our dataset. It is ready to be utilized in model building which will be done in the next section of this report.

Deciding Models and Model Building

In this section, we will discuss the implementation of various classifier models from Machine Learning, Neural Networks and LSTM.

ML classifiers are not commonly preferred in NLP applications as they lack the ability to comprehend the significance of words with respect to their relationship to surrounding words and sentences. This is because ML classifiers merely analyze the corpus word by word individually to form patterns to learn from. As our independent feature is a free form textual description of an event that has occured, it may not prove to be suited for ML classification techniques. However, as a preliminary model, we have gone ahead with implementing the ML classification.

Neural Network models are better suited to approach NLP classification problems and hence, we have higher expectations from a NN.

Lastly, LSTM is expected to be the best performing model as it is aptly suited for implementation on context based classification where there needs to be a high emphasis on sequentially learning the patterns underlying in the corpus.

We have 2 target features : 'Accident Level' and 'Potential Accident Level'. Whereas the 'Description' column is our independent feature. So, we will be keeping the model architecture the same and just change the target features to get results for both.

Here we will define our target variable. Below is the code block showing the use of LabelBinarizer to fit and transform the target values. Also, this is to apply the One hot encoding.

```
from sklearn.preprocessing import LabelBinarizer
lb = LabelBinarizer()
Y = lb.fit_transform(data['Accident Level'].values)
Y
```

```
from sklearn.preprocessing import LabelBinarizer
lb = LabelBinarizer()
Y = lb.fit_transform(data['Potential Accident Level'].values)
Y
```

Fig 14: Code for LabelBinarizer for both Targets

Data Splitting

Now, we can proceed with splitting our data into train and test sets as shown below.

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.1,random_state=1)
```

Fig 15: Code for Train/Test Split

Machine Learning Classifier

We will pick the Logistic Regression Model, Random Forest Classifier and SVM to conduct the classification for both targets.

LogisticRegression

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression

lmodel = LogisticRegression(solver='liblinear')
lmodel = OneVsRestClassifier(lmodel)
```

```
lmodel.fit(x_train,y_train)
```

```
predicted_labels = lmodel.predict(x_test)
predicted_scores = lmodel.decision_function(x_test)
```

With Target = 'Accident Level'

```
from sklearn.metrics import accuracy_score
print('Test data Accuracy score: ', accuracy_score(y_test, predicted_labels))
```

```
Test data Accuracy score:  0.6352941176470588
```

```
print('Training data Accuracy score: ',lmodel.score(x_train,y_train))
```

```
Training data Accuracy score:  0.7676470588235295
```

With Target = 'Potential Accident Level'

```
from sklearn.metrics import accuracy_score
print('Test data Accuracy score: ', accuracy_score(y_test, predicted_labels))
```

```
Test data Accuracy score:  0.12941176470588237
```

```
print('Training data Accuracy score: ',lmodel.score(x_train,y_train))
```

```
Training data Accuracy score:  0.3735294117647059
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 42)
classifier.fit(x_train, y_train)
```

```
y_predr = classifier.predict(x_test)
print('Test data Accuracy score: ', accuracy_score(y_test, y_predr))
```

With Target = 'Accident Level'

```
y_predr = classifier.predict(x_test)
print('Test data Accuracy score: ', accuracy_score(y_test, y_predr))
```

```
Test data Accuracy score:  0.7058823529411765
```

```
print('Train data Accuracy score: ', classifier.score(x_train, y_train))
```

```
Train data Accuracy score:  0.9294117647058824
```

With Target = 'Potential Accident Level'

```
y_predr = classifier.predict(x_test)
print('Test data Accuracy score: ', accuracy_score(y_test, y_predr))
```

```
Test data Accuracy score:  0.12941176470588237
```

```
print('Train data Accuracy score: ', classifier.score(x_train, y_train))
```

```
Train data Accuracy score:  0.9
```

SVM

As a last ML model, we will deploy SVM to check if it performs better than the previous models.

```
from sklearn import svm
from sklearn.multiclass import OneVsRestClassifier
clf = svm.SVC(kernel='poly',degree=5, C=1)
clf = OneVsRestClassifier(clf)

clf.fit(x_train , y_train) # Fit the model
```

With Target = 'Accident Level'

```
y_pred = clf.predict(x_test)
print('Test data Accuracy score: ', accuracy_score(y_test, y_pred))

Test data Accuracy score:  0.6941176470588235

print('Training data Accuracy score: ',clf.score(x_train,y_train))

Training data Accuracy score:  0.9117647058823529
```

With Target = 'Potential Accident Level'

```
y_pred = clf.predict(x_test)
print('Test data Accuracy score: ', accuracy_score(y_test, y_pred))

Test data Accuracy score:  0.07058823529411765

print('Training data Accuracy score: ',clf.score(x_train,y_train))

Training data Accuracy score:  0.5911764705882353
```

Neural Network Classifier

Next, we will implement a Neural Network for both the Target features and test them individually.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten,Dropout
from tensorflow.keras import regularizers, optimizers

# Model architecture
modelNN = Sequential()
modelNN.add(Dense(512, input_dim=maxlen, kernel_initializer='normal', activation='relu'))
modelNN.add(Dense(256, kernel_initializer='normal',activation='relu'))
modelNN.add(Dropout(0.5))
modelNN.add(Flatten())
modelNN.add(Dense(128,activation='relu'))
modelNN.add(Dense(64,activation='relu'))
modelNN.add(Dropout(0.3))
modelNN.add(Dense(5,activation='softmax'))

# Compile the model
modelNN.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

modelNN.fit(x_train, y_train, epochs=50, batch_size=32, verbose= 1,validation_split=0.1)
```

After compilation, we will fit and evaluate the model first for 'Accident Level' then for 'Potential Accident Level'.

With Target = 'Accident Level'

```
print('Training data Accuracy score: ',modelNN.evaluate(x_train,y_train,verbose=1))

11/11 [==============================] - 0s 2ms/step - loss: 0.2146 - accuracy: 0.7471
Training data Accuracy score:  [0.214633047580719, 0.7470588088035583]

print('Testing data Accuracy score: ',modelNN.evaluate(x_test,y_test,verbose=1))

3/3 [==============================] - 0s 4ms/step - loss: 0.5476 - accuracy: 0.7529
Testing data Accuracy score:  [0.5476469397544861, 0.7529411911964417]
```

With Target = 'Potential Accident Level'

```
print('Training data Accuracy score: ',modelNN.evaluate(x_train,y_train,verbose=1))

11/11 [==============================] - 0s 2ms/step - loss: 0.4056 - accuracy: 0.4324
Training data Accuracy score:  [0.4055951237678528, 0.4323529303073883]

print('Testing data Accuracy score: ',modelNN.evaluate(x_test,y_test,verbose=1))

3/3 [==============================] - 0s 3ms/step - loss: 0.4082 - accuracy: 0.2824
Testing data Accuracy score:  [0.4082416296005249, 0.2823529541492462]
```

LSTM Classifier

Lastly, we will build, compile, fit and evaluate the LSTM model for both the targets individually.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,Dropout,Bidirectional,Dense,Flatten,Add,Embedding

model_lstm = Sequential()
model_lstm.add(Embedding(num_words,embedding_size,weights=[embedding_matrix],input_length=maxlen))
model_lstm.add(Dropout(0.3))
model_lstm.add(Bidirectional(LSTM(452,dropout=0.2,recurrent_dropout=0.2)))
model_lstm.add(Flatten())
model_lstm.add(Dense(256,activation='relu'))
model_lstm.add(Dense(128,activation='relu'))
model_lstm.add(Dropout(0.5))
model_lstm.add(Dense(labels,activation='softmax'))

model_lstm.compile(optimizer='adam',metrics=['accuracy'],loss='binary_crossentropy')

model_lstm.fit(x_train,y_train,batch_size=32,epochs=10,verbose=1,validation_split=0.1)
```

With Target = 'Accident Level'

```
print('Training data Accuracy score: ',model_lstm.evaluate(x_train,y_train,verbose=1))

11/11 [==============================] - 5s 410ms/step - loss: 0.1273 - accuracy: 0.8941
Training data Accuracy score:  [0.12731054425239563, 0.8941176533699036]

print('Testing data Accuracy score: ',model_lstm.evaluate(x_test,y_test,verbose=1))

3/3 [==============================] - 1s 386ms/step - loss: 0.2981 - accuracy: 0.7765
Testing data Accuracy score:  [0.2980838418006897, 0.7764706015586853]
```

With Target = 'Potential Accident Level'

```
print('Training data Accuracy score: ',model_lstm.evaluate(x_train,y_train,verbose=1))

11/11 [==============================] - 0s 26ms/step - loss: 0.6621 - accuracy: 0.3235
Training data Accuracy score:  [0.6621089577674866, 0.3235294222831726]

print('Testing data Accuracy score: ',model_lstm.evaluate(x_test,y_test,verbose=1))

3/3 [==============================] - 0s 27ms/step - loss: 0.6620 - accuracy: 0.3882
Testing data Accuracy score:  [0.6620379090309143, 0.38823530077934265]
```

Model Performance

Model Accuracies

We will now evaluate all the models we have built and implemented. To do so, firstly, we will collaborate all the accuracy results respective to each model in the table below.

The below data is taken from the last run of the submitted notebook

Preprocessing Method	'AL' MODELS	TRAIN			TEST		
		ACC%	RECALL	ROC	ACC%	RECALL	ROC
GloVE	LogisticRegression	76.1			65.8		
	RandomForest	93.2			72.9		
	SVM	90.8			68.2		
	NeuralNetwork	74.1			75.3		
	LSTM	86.8			72.9		
TF-IDF	LogisticRegression	74.3			72.1		
	RandomForest	94.76			65.1		
	SVM	98.69			72.1		
	NeuralNetwork	97.91			67.4		
	LSTM	72.1			74.6		

This has been filled out properly in the 'Final Report Section'

Preprocessing Method	'PAL' MODELS	TRAIN			TEST		
		ACC%	RECALL	ROC	ACC%	RECALL	ROC
GloVE	LogisticRegression	37.4			12.9		
	RandomForest	86.8			16.4		
	SVM	58.8			5.8		
	NeuralNetwork	32.1			34.1		
	LSTM	74.7			49.4		
TF-IDF	LogisticRegression	13.6			7.0		
	RandomForest	88.2			11.6		
	SVM	99.4			9.3		
	NeuralNetwork	96.6			44.1		
	LSTM	37.2			33.2		

We can see that the models built for 'Accident Level' are significantly better performers both in terms of Train and Test accuracy. This result is expected as the 'Potential Accident Level' feature is not intrinsically predictable. As it is the potential rating of how severe the accident *could* have been, there is really no accurate way of reporting this without high subjectivity from the source of the data. We have discussed this earlier while discussing FIG 3 in the second section of this report.

We have used two kinds of preprocessing methods: GloVE and TF-IDF. We can see that the GloVE based models have better accuracy scores across all models.

From the obtained results, we have selected the LSTM Model for this classification problem as it has shown all round performance in both test and train accuracies. The ML models have been observed to be performing great in train data and fairly poor in test data. This is due to overfitting caused by lack of contextual understanding within the model architecture itself. Lastly, the NeuralNetwork models have also not performed as expected. Hence LSTM is the clear choice of model to proceed with.

Next Steps in our Project

We will summarize the steps we wish to take to carry this project forward and move towards implementing the Milestone 3 as well as making our final submission.

1. We will explore and implement methods for improving our model accuracies such as hyper-parameter tuning and featurization.
2. We will extract detailed metrics which describe the performance of our models in more depth such as Recall, Precision and ROC. This will help us to identify the most robust and best performing model.
3. Building the NLP Chatbot GUI .
    a. We have made progress in generating a pickle file.
    b. We aim to include at least 3 interfaces in the GUI as per Milestone 3.
    c. We will finalize the pipeline for the Chatbot deployment.

Final Report Section

The salient addition that we have made to our interim project is the addition of some ensemble techniques like bagging and boosting, collecting the results of each model in a separate data frame and printing this as a table for metrics comparison. We have also added metrics such as accuracy, F1 score, precision, recall and roc_auc. This helps us in a broader understanding of the performance of the models. Lastly, we have performed Cross Validation of our ML models accuracy projection as a box plot to get an in depth idea of performance which is not completely reliant on just the accuracy.

Below are the visuals depicting what is discussed above:

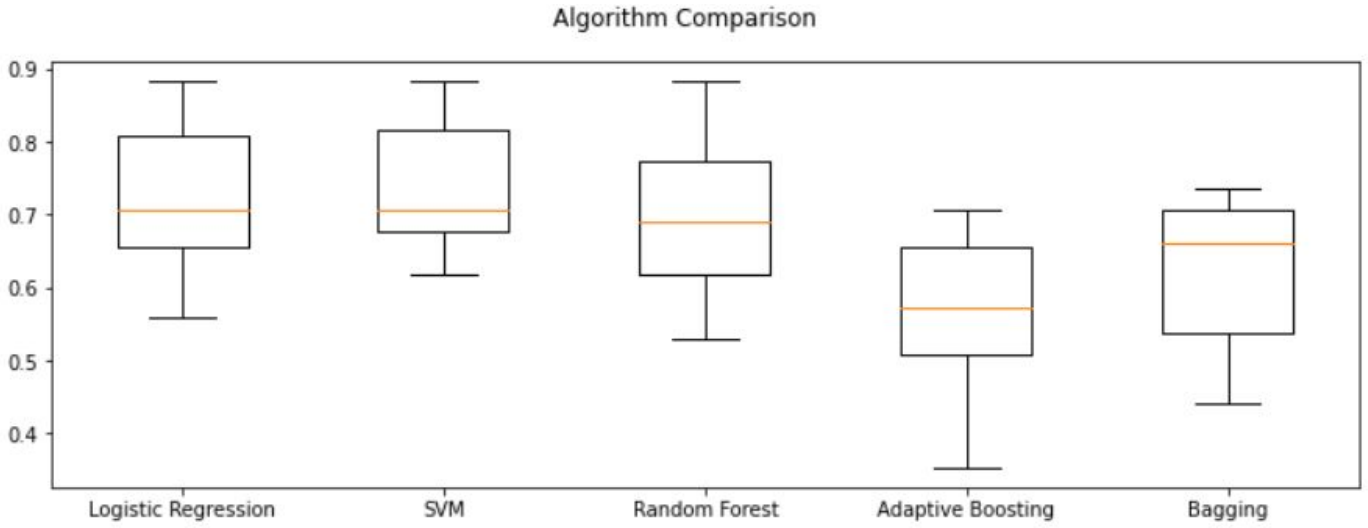
Training Metrics Accuracy Of AL

	Model	accuracy	F1 score	precision	recall	roc_auc
1	Logistic Regression	0.782353	0.805882	0.68827	0.805882	0.878676
2	Random Forest	0.926471	0.951662	0.921036	0.926471	0.960662
3	SVM	0.911765	0.913108	0.851414	0.911765	0.945221
5	Bagging	0.988235	0.99115	0.984741	0.988235	0.993382
4	Boosting	0.782353	0.837209	0.731602	0.847059	0.901471
6	NN	0.726471	0.678039	0.635662	0.726471	0.902117
7	LSTM	0.779412	0.761332	0.852957	0.779412	0.958946

Testing Metrics Accuracy Of AL

	Model	accuracy	F1 score	precision	recall	roc_auc
1	Logistic Regression	0.782353	0.805882	0.68827	0.805882	0.878676
2	Random Forest	0.926471	0.951662	0.921036	0.926471	0.960662
3	SVM	0.911765	0.913108	0.851414	0.911765	0.945221
5	Bagging	0.988235	0.99115	0.984741	0.988235	0.993382
4	Boosting	0.782353	0.837209	0.731602	0.847059	0.901471
6	NN	0.726471	0.678039	0.635662	0.726471	0.902117
7	LSTM	0.779412	0.761332	0.852957	0.779412	0.958946

Cross Validation and ML models accuracy projection as box plot.



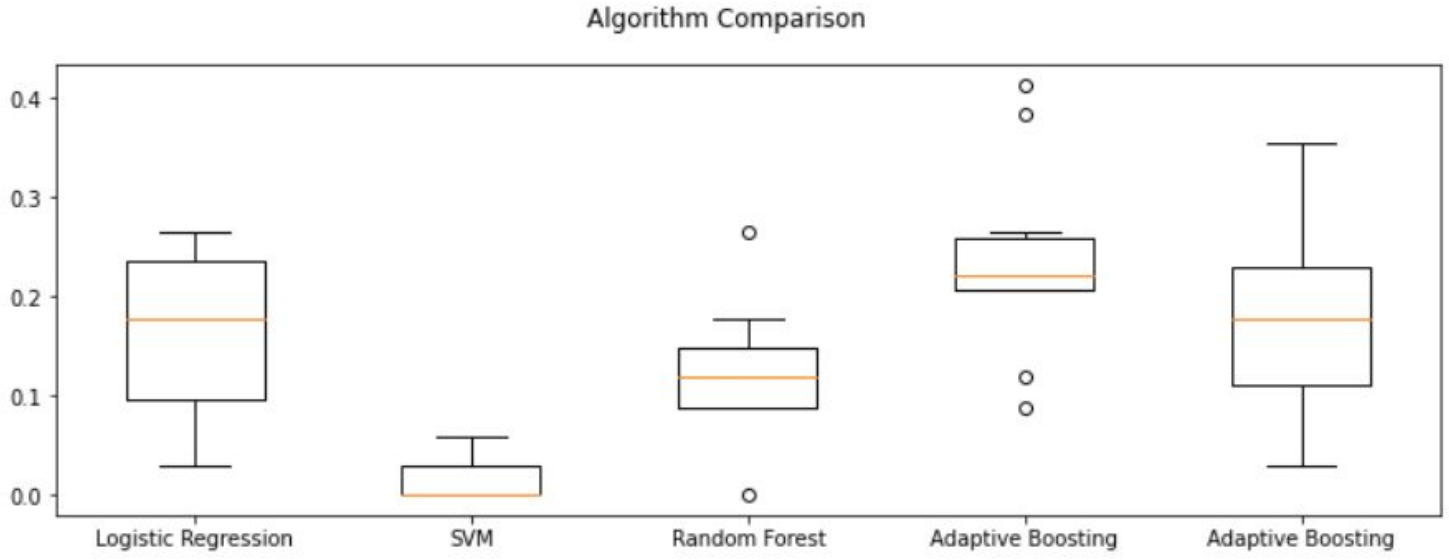
Training Metrics Accuracy Of PAL

	Model	accuracy	F1 score	precision	recall	roc_auc
1	Logistic Regression	0.364706	0.500935	0.4223	0.394118	0.667157
2	Random Forest	0.905882	0.949153	0.92648	0.905882	0.952451
3	SVM	0.591176	0.743068	0.693382	0.591176	0.795588
5	Bagging	0.994118	0.99705	0.995588	0.994118	0.997059
4	Boosting	0.441176	0.594697	0.520181	0.461765	0.715686
6	NN	0.241176	0.168514	0.129499	0.241176	0.600702
7	LSTM	0.826471	0.837108	0.856888	0.826471	0.961672

Test Metrics Accuracy Of PAL

	Model	accuracy	F1 score	precision	recall	roc_auc
1	Logistic Regression	0.141176	0.2	0.255948	0.152941	0.513725
2	Random Forest	0.211765	0.318584	0.333193	0.211765	0.586275
3	SVM	0.0823529	0.225806	0.267949	0.164706	0.533333
5	Bagging	0.994118	0.99705	0.995588	0.994118	0.997059
4	Boosting	0.235294	0.353846	0.320654	0.270588	0.592157
6	NN	0.164706	0.261339	0.529563	0.176471	0.539571
7	LSTM	0.564706	0.610457	0.686957	0.564706	0.776458

Cross Validation and ML models accuracy projection as box plot.



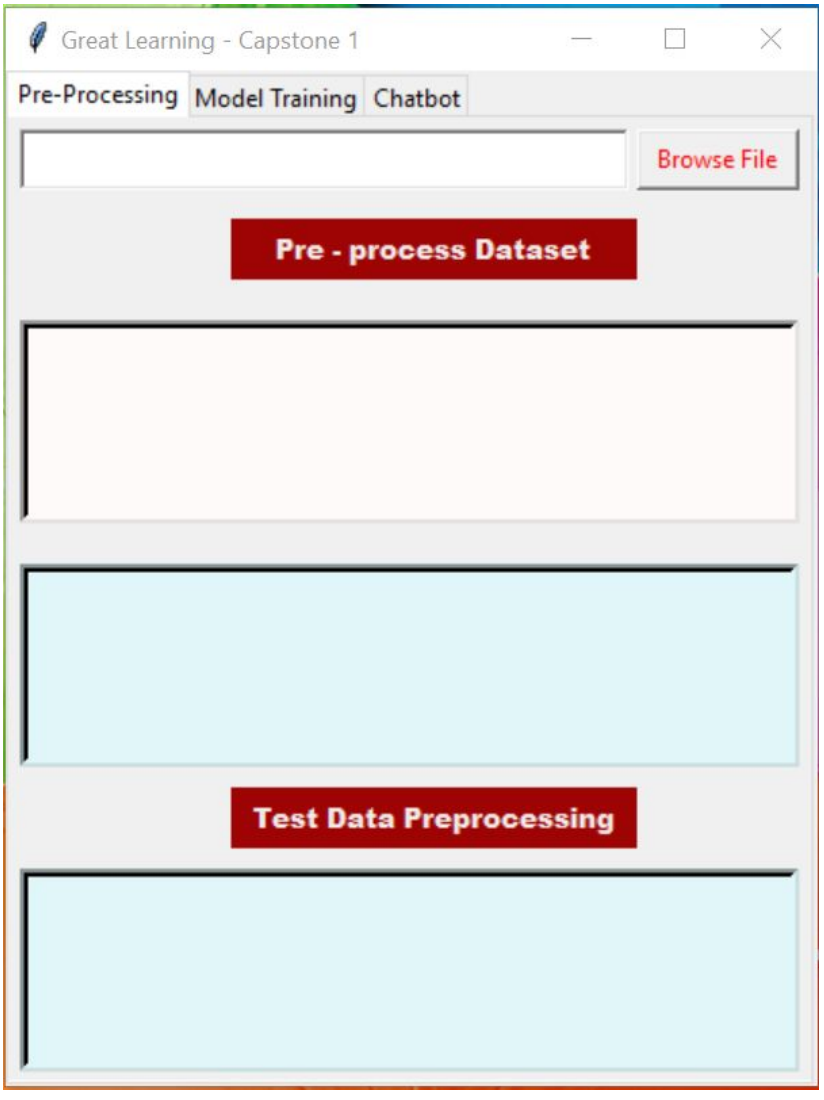
Model Selection

We can clearly conclude that LSTM is the best model to be implemented in the chatbot for the same reasons as discussed earlier.

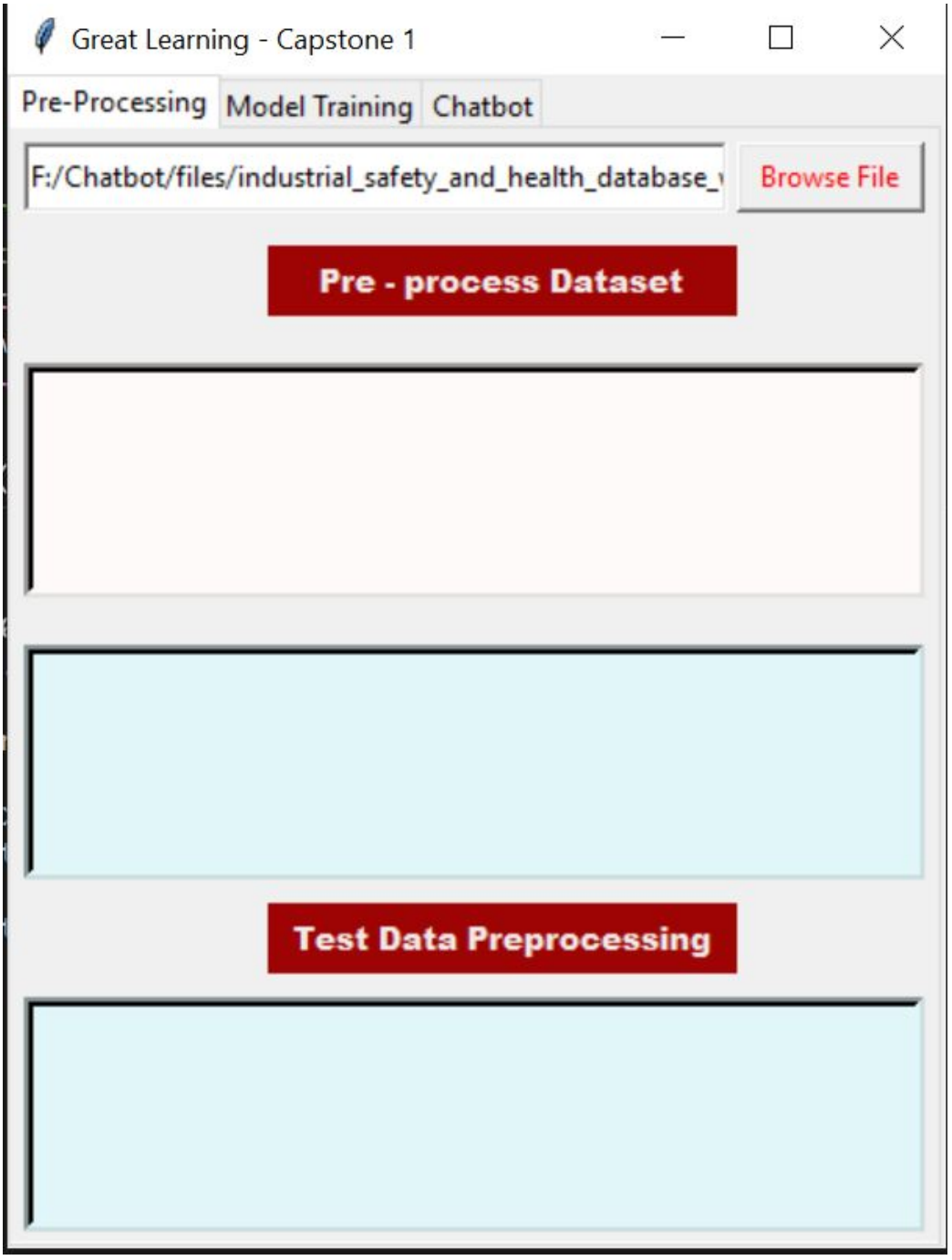
CHATBOT DEVELOPMENT

To successfully complete the Milestone 3 of our capstone project, we used the selected models in the previous sections to process the text data in a chat window format with 3 buttons on the GUI. The function of the 3 buttons was

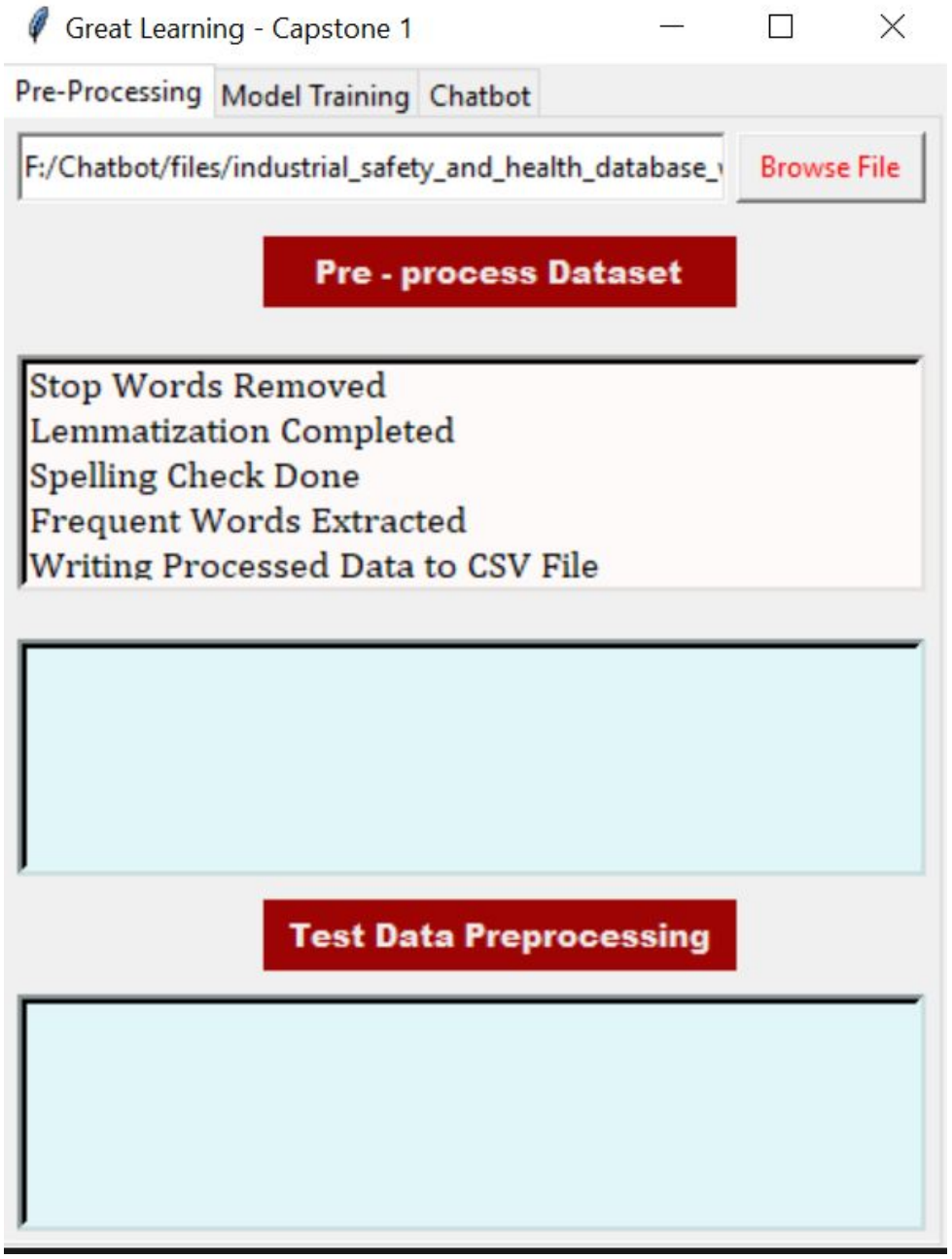
1. Button 1 : Chatbot Interface Screen
2. Button 2 : Preprocessing
3. Button 3 : Predicting using selected Model



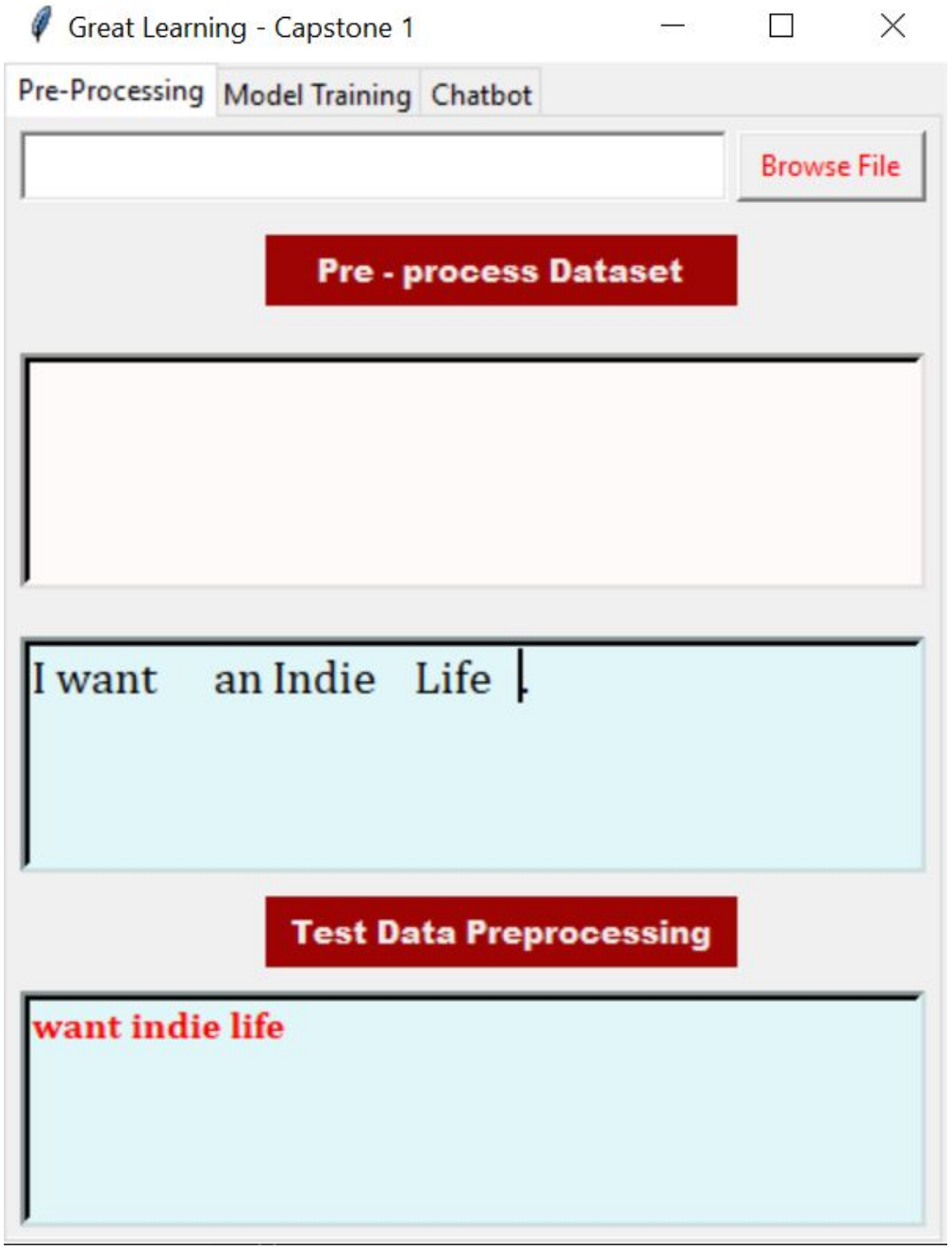
This is the Main Screen of the Chatbot UI along with the 3 buttons shown as tabs.



Here we are loading the dataset using the 'Browse File' inbuilt button. This helps us to run the Chatbot locally on our system. This button will preprocess the whole dataset and generate an output file which will be used for training future input.

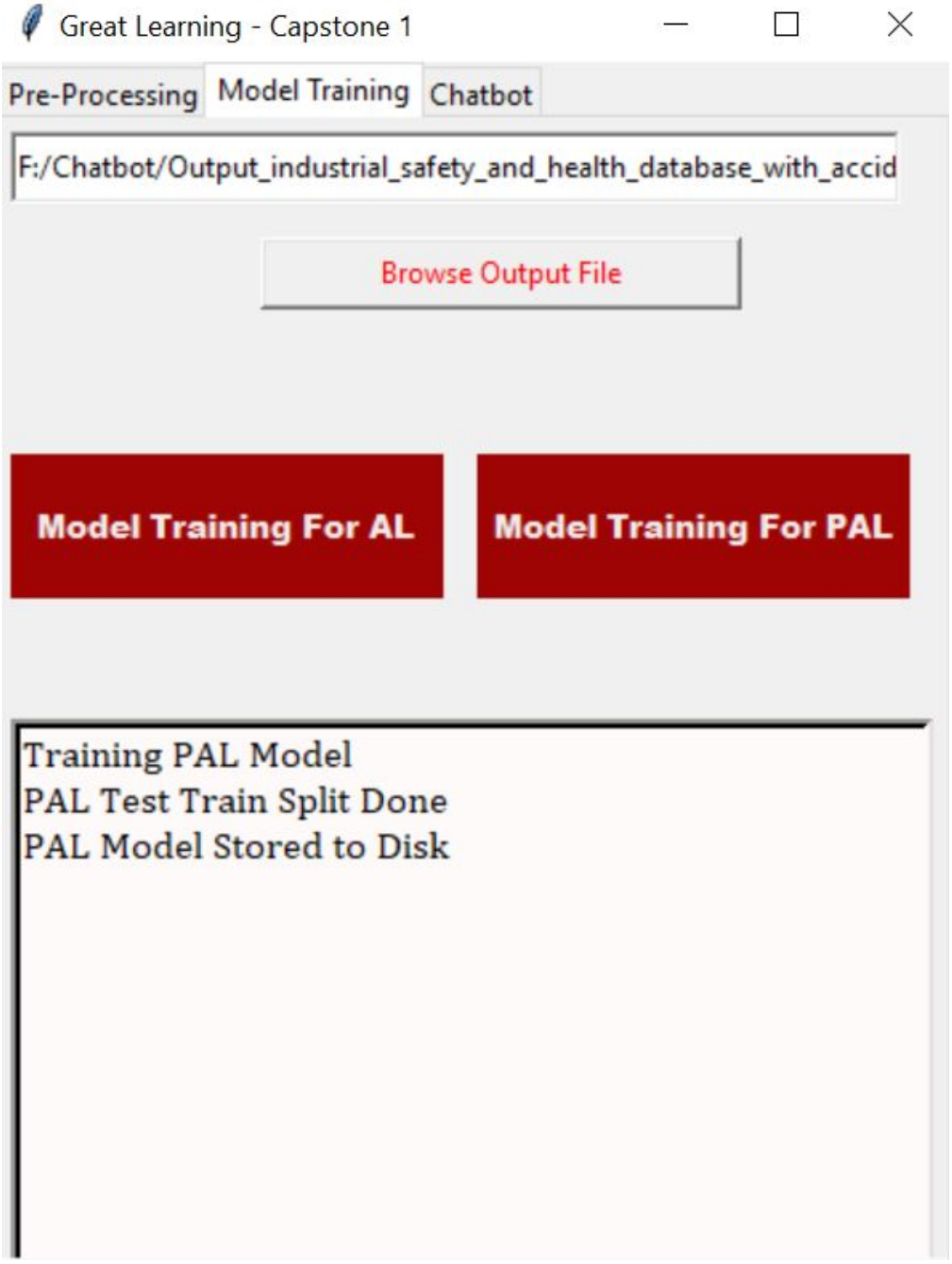


Here we are running the Preprocessing on the entire Dataset. It is showing the various functions that are being run in the background like removing StopWords, Lemmatization, Spell Check etc. The output is then saved as a CSV file.

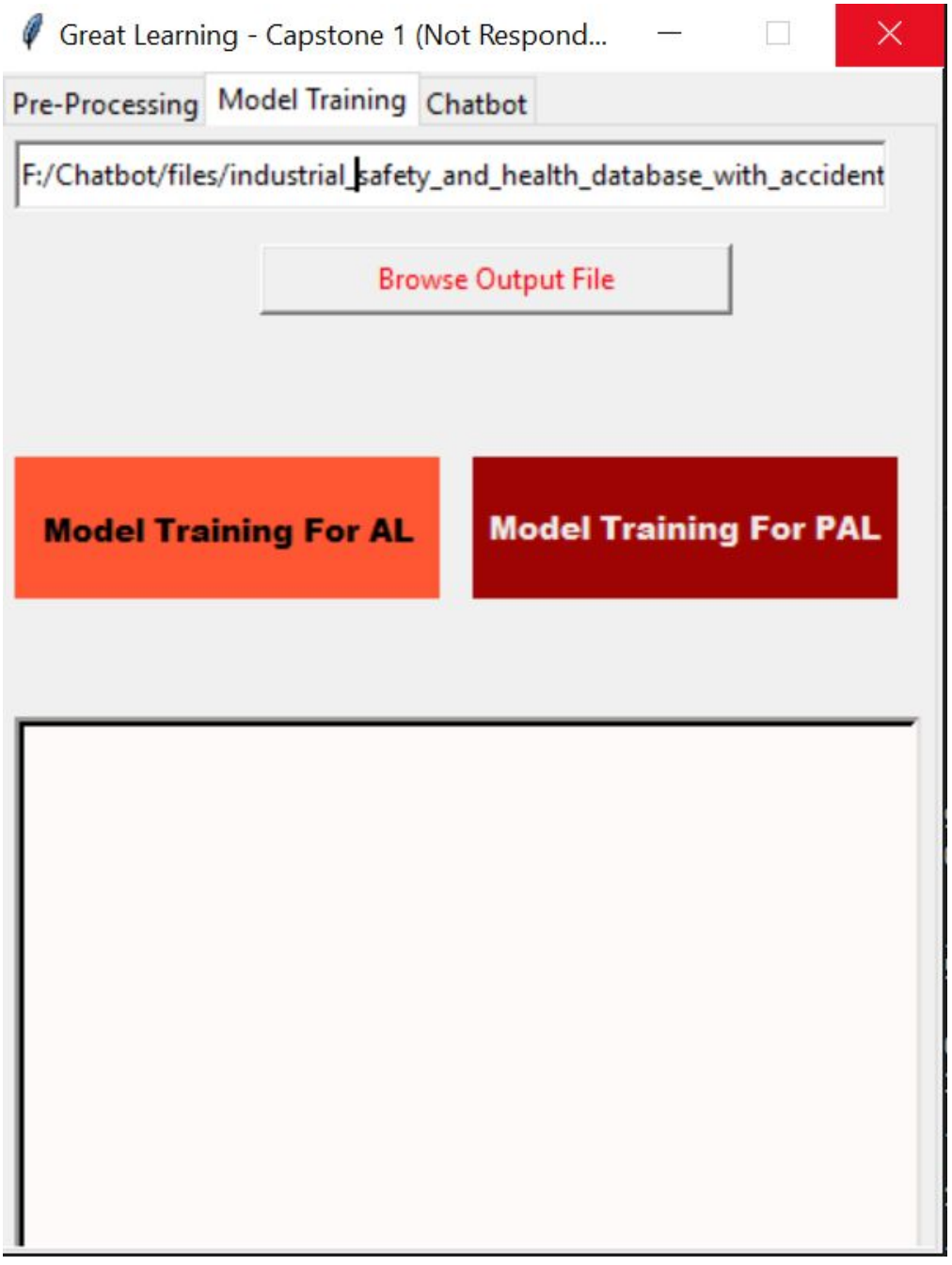


Here we have typed in a random text using the chatbot and run the preprocessing function on the input text. The processed output is displayed in the below window.

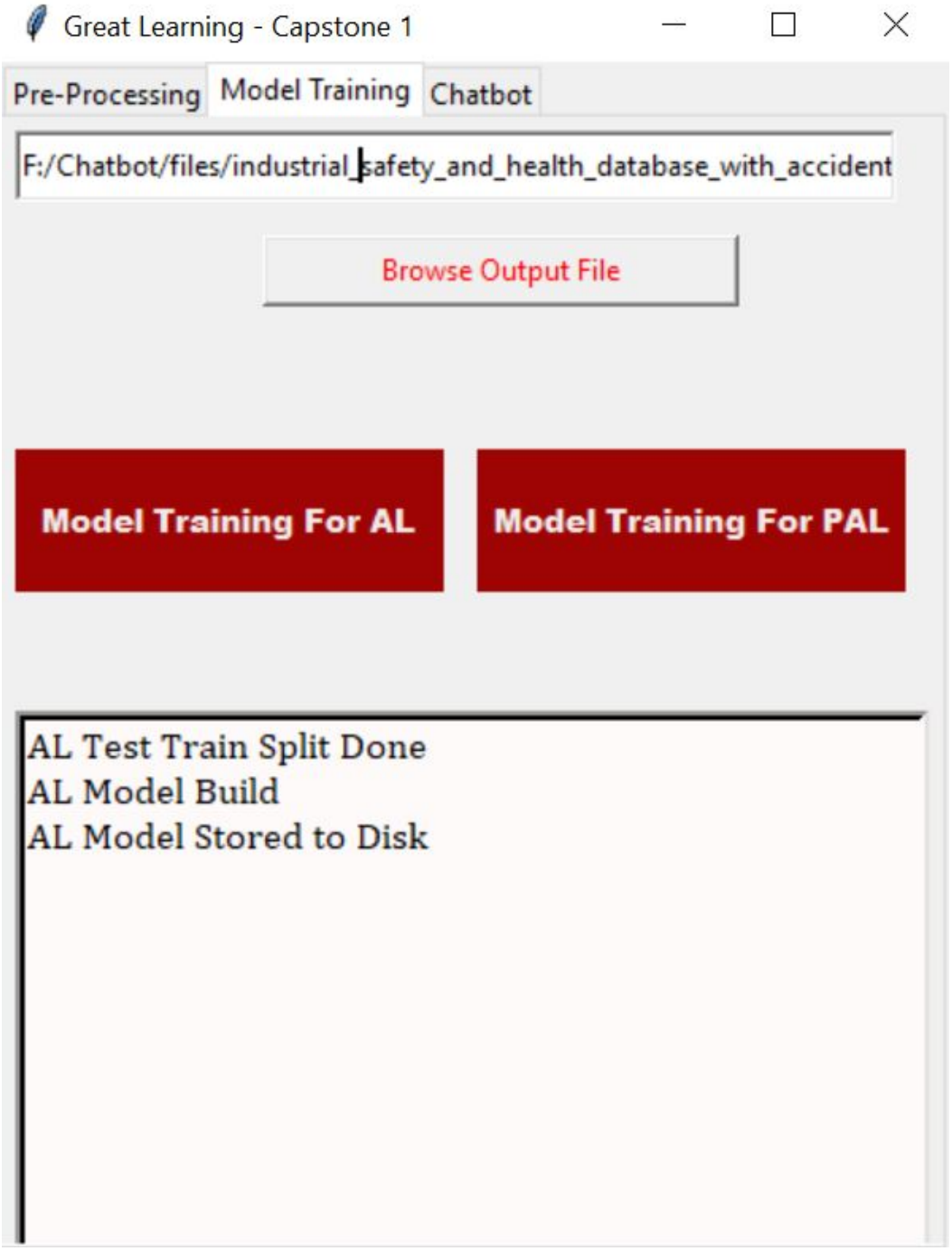
This is the view of the Model Training button - here we have the bifurcation for Accident Level and Potential Accident Level which are the two outputs required. Here it will run for PAL and later we will discuss for AL.



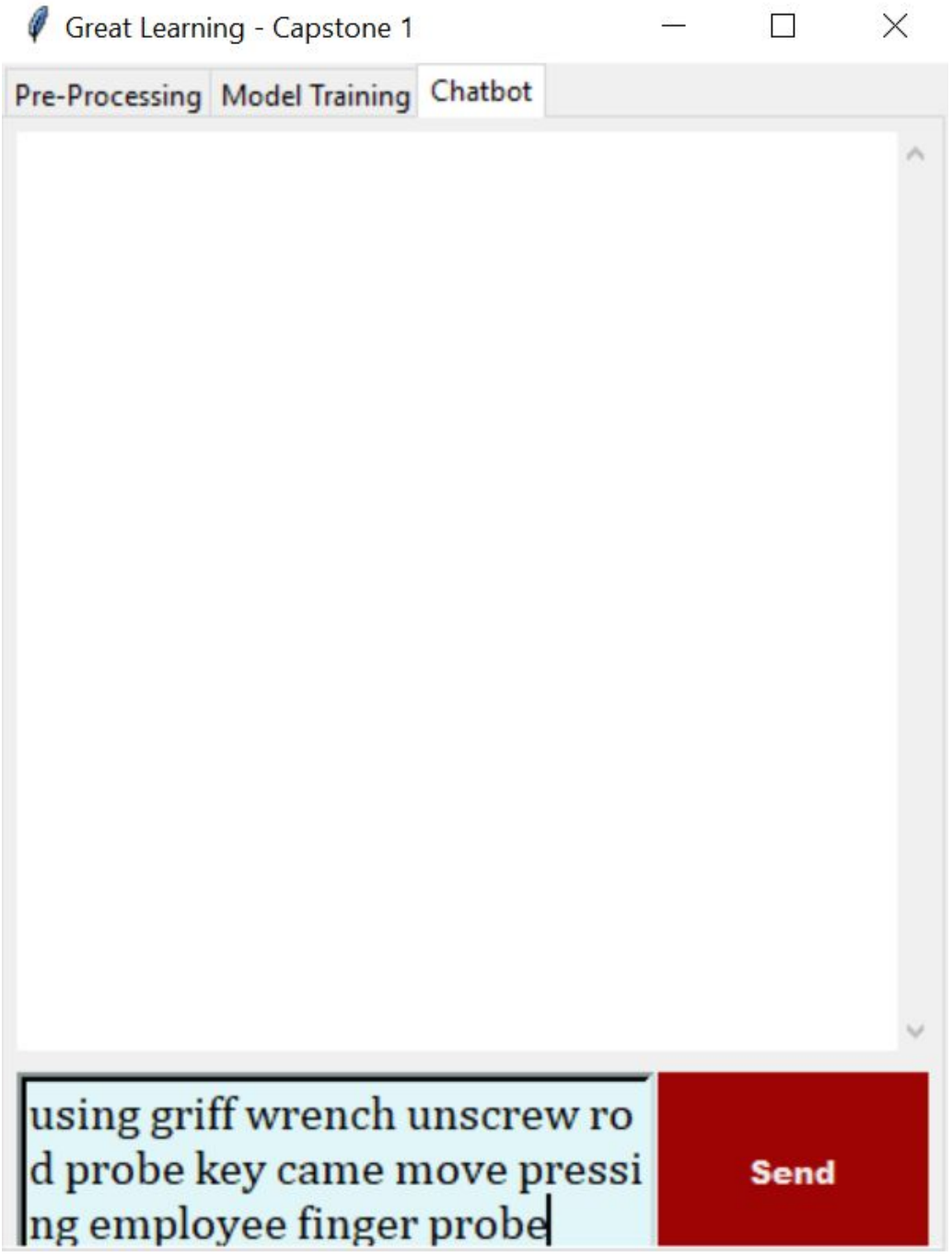
We can see the chatbot is performing the training on PAL here as well as splitting the data set and storing the model locally.



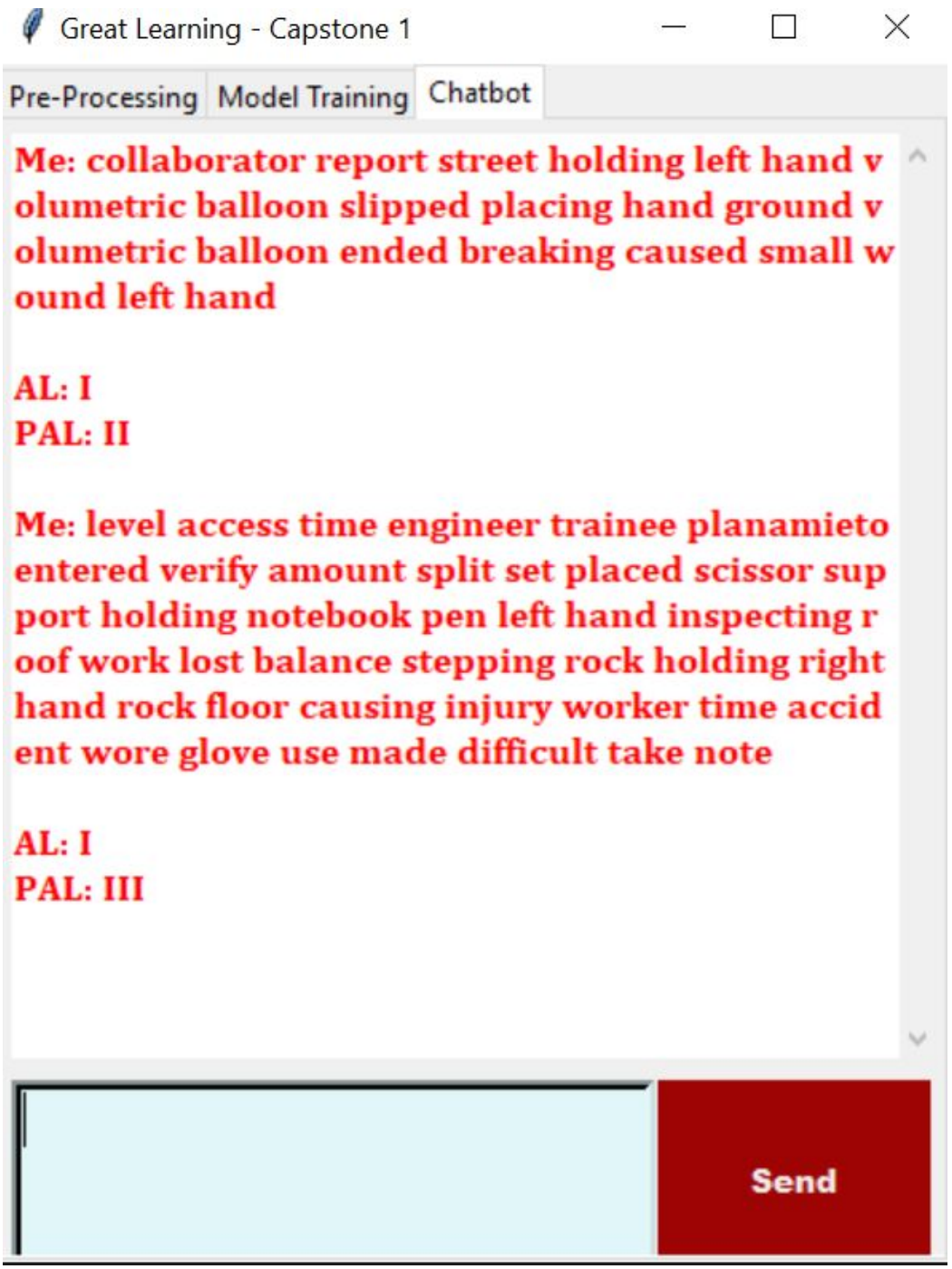
We can see that here to Chatbot is training for AL.



Similarly, the trained model for AL is stored locally.



Finally, we are now typing out a description in the Chatbot button tab. This will be the input data for our models.



Here is the output of the chatbot where it is predicting the Accident level and Potential Accident Level based on the description of the accident.


Great Learning - Capstone 1
Pre-Processing
Model Training
Chatbot
Me: using griff wrench unscrew rod probe key ca me move pressing employee finger probe
AL: II
PAL: III
Send

Discussion:

We can conclude that we have successfully built a chatbot interface that can analyze a given text input and predict the required outcomes based on this.

Implications, limitations and Improvements

However, there is much room for improvement. Here are few points that were discussed within the team:

1. The chatbot can be made to be more human-like with some opening statements like 'Welcome! I am your accident assistant. Please describe in details what has happened and I will help you to analyze the Accident Level as well as the Potential Accident Level' and intermittent statements like 'Please give me a moment, I am analyzing' etc
2. There would need to be rules added to the Chatbot such as minimum length of input to be accepted so that a simple 'Hi' or 'Hello' is not considered as input. This would also eliminate poor predictions as a certain length of description is required for accuracy. In the case where this rule is not met, we can have an error statement such as 'Sorry, I would need more detail to help you out. Please elaborate'
3. Hosting the GUI over a web based service for easier access as well as faster interactions as the training and testing data can be stored there itself and there wouldn't be a need to run it over locally everytime.

Closing Reflections

This project was a great opportunity to dive deep into NLP. From the get go, we had to revisit and bring in concepts from the beginning of the course until the end. A comprehensive exercise such as this helped us to understand team building and team work in the field of AIML. All participants went above and beyond to tick off the tasks that were self assigned.

Moreover, we had immense support from an insightful mentor - Sravan Malla, who helped us at every step of the way. We appreciate the fact that his style of teaching left us seeking answers on our own and built confidence in our approach to problem solving.

Lastly, we wholeheartedly commend the Greatlearning support team and our Program Manager - Divya Teresa, who has been the go to person for any and every requirements we've had from day one in this course.

We are glad to have had this Greatlearning experience and will certainly cherish and utilize the knowledge we have picked up here to help our community, to bring solutions and to make the world a better place.

Link to Dataset

Link to download the dataset:
https://www.kaggle.com/ihmstefanini/industrial-safety-and-health-analytics-database

References

[1] Boss Bot: Your Guide to Talking Chatbots | Smartsheet
https://www.smartsheet.com/artificial-intelligence-chatbots

[2] NLTK's list of english stopwords
https://gist.github.com/sebleier/554280

[3] Lemmatization Approaches with Examples in Python
https://www.machinelearningplus.com/nlp/lemmatization-examples-python/#wordnetlemmatizer