Final Report for Capstone Project

NLP based Chatbot on Industrial Safety.

Date-24-07-2022

Group 11 Jul 21A NLP Project 2:

Submitted By:-
1. Bhavadharini Parthasarathy
2. Keerthana V Elango
3. Surajit Pal
4. Vaibhavi Gujar
5. Prajual Gupta
6. Prachi Sangram Pawar

- Domain: NLP based Chatbot on Industrial Safety.

- Context: The database comes from one of the biggest industry 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. Sometimes they also die in such environment.

- DATA DESCRIPTION: This The database is basically records of accidents from 12 different plants in 03 different countries which every line in the data is an occurrence of an accident.

- 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 VI, it registers how severe was the accident (I means not severe but VI means very severe)
  - Potential Accident Level: Depending on the Accident Level, the database also registers how severe the accident could have been (due to other factors involved in the accident)
  - Gender: if the person is male of 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.

Load And Import Data:

	Data	Countries	Local	Industry Sector	Accident Level	Potential Accident Level	Genre	Employee or Third Party	Critical Risk	Description
	2016-01-01 00:00:00	Country_01	Local_01	Mining	I	IV	Male	Third Party	Pressed	While removing the drill rod of the Jumbo 08 f...
	2016-01-02 00:00:00	Country_02	Local_02	Mining	I	IV	Male	Employee	Pressurized Systems	During the activation of a sodium sulphide pum...
	2016-01-06 00:00:00	Country_01	Local_03	Mining	I	III	Male	Third Party (Remote)	Manual Tools	In the sub-station MILPO located at level +170...
	2016-01-08 00:00:00	Country_01	Local_04	Mining	I	I	Male	Third Party	Others	Being 9:45 am. approximately in the Nv. 1880 C...
	2016-01-10 00:00:00	Country_01	Local_04	Mining	IV	IV	Male	Third Party	Others	Approximately at 11:45 a.m. in circumstances t...

Data Cleansing Summary:

- We are left with 418 rows and 10 columns after data cleansing.
- Label Encoding Changing the Column Accident level And Potential Accident level to countable as they were in roman numbers .
- Removed 'Unnamed: 0' column and renamed - 'Data', 'Countries', 'Genre', 'Employee or Third Party' and 'Accident Level' columns in the dataset.
- We had 7 duplicate instances in the dataset and dropped those duplicates.
- There are no outliers in the dataset.
- No missing values in dataset.

EDA Data Analysis And Visualization Summary:

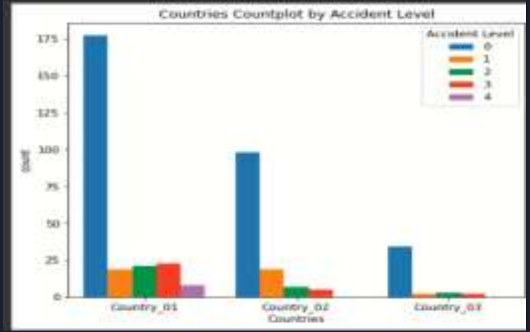
Univariate Analysis Observation :

- ★ The most affected country from the above dataset is country_01 with around 59% of the accidents with the count of 250.
- ★ Most accidents happened in Local_03 .Its count is 90 ,which is equivalent to 21.18%. The second Most Accident happens in local_5 which is equivalent to 13.88%.
- ★ Mostly affected sector is Mining sector. 56.71% of accidents occur in Mining sector.
- ★ Most affected workers in accidents are male .Their count is 403 ,which is equivalent to 94.82%.
- ★ Most accidents belongs to "Accident Level - I" .Its count is 316 which is equivalent to 74.35%% of total accidents.
- ★ Most "Potential Accident Level" belongs to level IV .Its count is 143 which is equivalent to 33.65% of total potential accidents.
- ★ Most affected Employee type are Third party workers .Their count is 189 ,which is equivalent to 44.47%.
- ★ Most accidents happen in year 2016.Count is 285 ,which is equivalent to 67.06% .
- ★ Most accidents happen in Feb month.Count is 61 ,which is equivalent to 14.35%.

Bivariate Analysis Observation:

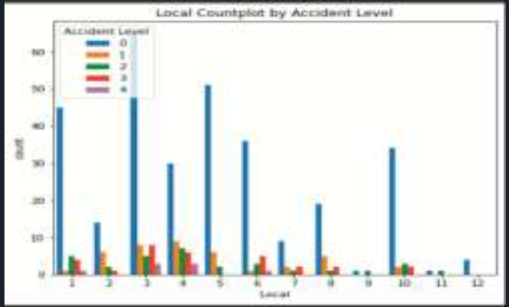
- ★ *Country Vs Accident Level*

  Accident level I is highest in all countries.
  Most accidents happened in Country_01.
  Accident level in Country_03 is lesser than other countries.



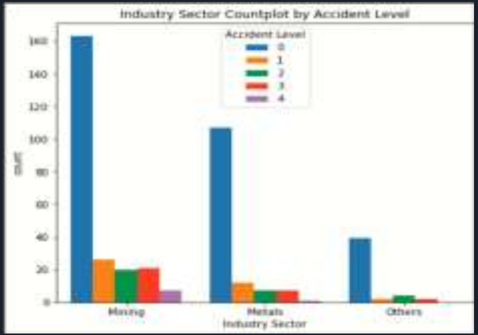
★ *Local Vs Accident Level*

Accident level I is highest in almost all localities.
Accident level I is highest in Local 3.
Local 9,11 and 12 have less accidents level.



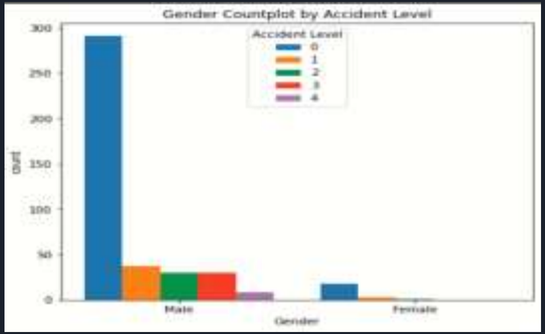
★ *Industry Sector Vs Accident Level*

Accident level I is highest in all industry sector (Mining, Metals and Other).
Most accidents happened in Mining industry sector.
After Accident Level I ,Level II is Highest among all the Industries.
There are very few cases for Accident level 5.



★ *Gender Vs Accident Level*

Accident level I is highest among the Gender.
Most accidents happened with male ones.
There are very few cases With Females.



DATA PRE- PROCESSING

- Lower Case : : Converting a word to lowercase. THe scaling of words will get easy. Its is pre processing technique

- Remove Punctuation: An important NLP preprocessing step is punctuation marks removal, this marks - used to divide text into sentences, paragraphs and phrases - affects the results of any text processing approach, especially what depends on the occurrence frequencies of words and phrases, since the punctuation marks are used frequently .

- Remove Stopwords : The words which are generally filtered out before processing a natural language are called stop words. These are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) and does not add much information to the text.

- ★ <u>Lemmatization</u>: Lemmatization is a text normalization technique . Essentially, lemmatization is a technique that switches any kind of a word to its base root mode.

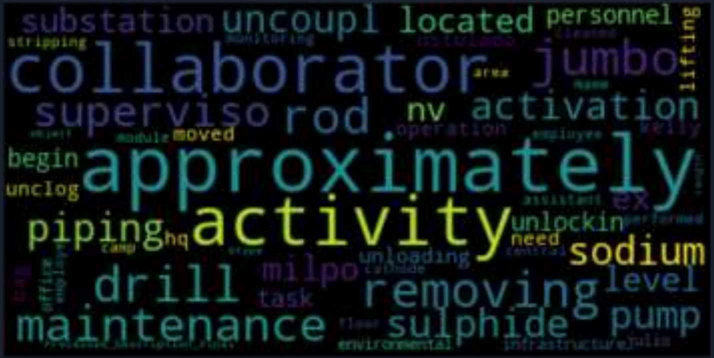
- ★ <u>Remove Extra Spaces :</u> Well, removing the extra space is good as it doesn’t store extra memory and even we can see the data clearly.

- ★ <u>Tokenization and keeping alphabets :</u> Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

- ★ <u>Word CLoud:</u> There are many body-related, employee related, movement-related, equipment-related and accident-related words.

★ <u>Word CLoud:</u> This is the word Cloud for different Accidental Level.

ACCIDENT LEVEL - 1

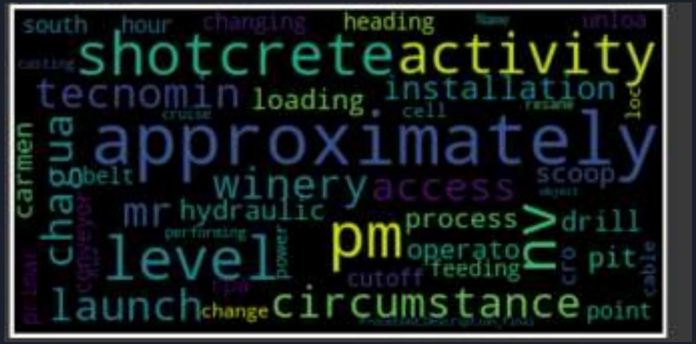


ACCIDENT LEVEL - 2

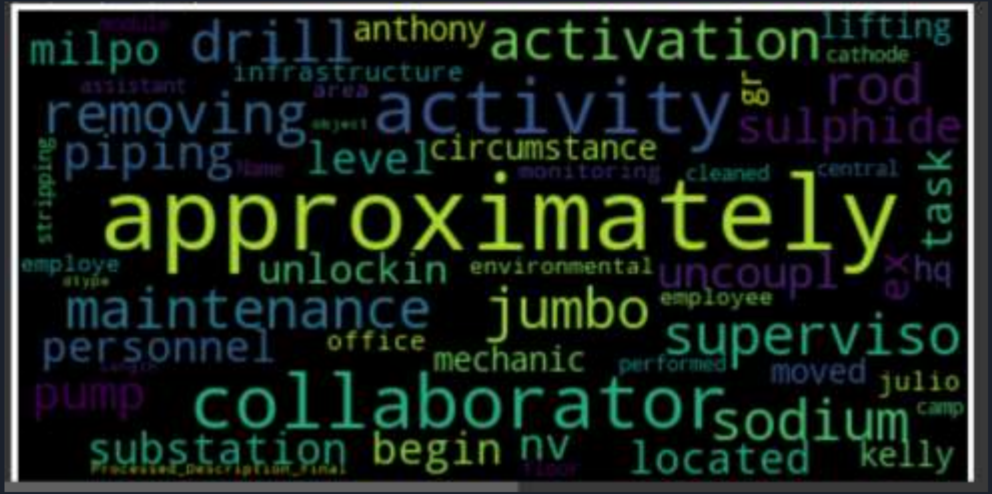
ACCIDENT LEVEL - 3

ACCIDENT LEVEL - 4

ACCIDENT LEVEL - 5



OVERALL WORD CLOUD



Feature Engineering :

- ★ Bag OF words : It shows how many times a word is occurred in the Document.

- ★ TF-IDF : which stands for term frequency—inverse document frequency, is a scoring measure widely used in information retrieval (IR) or summarization. TF-IDF is intended to reflect how relevant a term is in a given document.

- ★ Word2Vec Embeddings: This tool(Word2Vec) provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. The word2vec tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words. Converting the words back to the sentence form for modelling

- ★ Skipgram: Skip-gram is used to predict the context word for a given target word.Converting the words back to the sentence form for modelling

- ★ Fasttext: FastText is a library created by the Facebook Research Team for efficient learning of word representations and sentence classification

Create Training and Test Set:

1. Shape of Train Data x_train, y-train =334
2. Shape of Test Data x_test, y_test =84

Getting Target variable

```
X= df['Processed_Description_Final']
```

```
y = df['Accident Level']
```

Split data into train and test.

```
# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=42,test_size = 0.2)
```

```
print(X_train.shape)
print(y_train.shape)
```

```
(334,)
(334,)
```

```
print(X_test.shape)
print(y_test.shape)
```

```
(84,)
(84,)
```

Design, train and test machine learning classifiers
Build classifier Models.

- ★ Using BOW Vectorizer :

KNN performs best that all
other models because it is not overfitting
And the Test accuracy increases.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.757485	0.714286	0.682639	0.629371	0.757485	0.714286	0.756958	0.562500
1	SVM	0.766467	0.750000	0.688916	0.642857	0.766467	0.750000	0.804724	0.562500
2	Naive Bayes	0.215569	0.142857	0.231647	0.182104	0.215569	0.142857	0.760006	0.612566
3	KNN	0.736527	0.726190	0.635152	0.631034	0.736527	0.726190	0.625422	0.557927
4	Random Forest	0.994012	0.714286	0.993950	0.625000	0.994012	0.714286	0.994149	0.555556
5	Bagging	0.967066	0.666667	0.966122	0.604317	0.967066	0.666667	0.968036	0.552632
6	AdaBoost	0.577844	0.595238	0.577502	0.594234	0.577844	0.595238	0.584227	0.669389
7	Gradient Boost	0.859281	0.678571	0.839777	0.606383	0.859281	0.678571	0.877186	0.548077
8	XGBoost	0.868263	0.726190	0.852249	0.631034	0.868263	0.726190	0.883645	0.557927

Hyperparameter tuning for BOW

As we can see overfitting is being improved
and still KNN performs Better.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
1	SVM	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
2	Naive Bayes	0.224551	0.142857	0.234964	0.101982	0.495525	0.135714	0.355292	0.204517
3	KNN	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
4	Random Forest	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
5	Bagging	0.742515	0.750000	0.207382	0.171429	0.219094	0.200000	0.382724	0.150000
6	AdaBoost	0.730539	0.738095	0.168858	0.169663	0.198374	0.196825	0.146868	0.148398
7	Gradient Boost	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
8	XGBoost	0.742515	0.750000	0.197204	0.171429	0.214815	0.200000	0.281974	0.150000

★ Using TF-IDF Vectorizer:

Logistic regression
and kNN performs better.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
1	SVM	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
2	Naive Bayes	0.299401	0.166667	0.347516	0.225764	0.299401	0.166667	0.761703	0.559576
3	KNN	0.751497	0.690476	0.669087	0.612676	0.751497	0.690476	0.696138	0.550633
4	Random Forest	0.991018	0.738095	0.990841	0.836986	0.991018	0.738095	0.991191	0.560241
5	Bagging	0.961078	0.714286	0.958594	0.625000	0.961078	0.714286	0.962613	0.555556
6	AdaBoost	0.742515	0.702381	0.643664	0.618881	0.742515	0.702381	0.599551	0.553125
7	Gradient Boost	0.865269	0.678571	0.842212	0.610714	0.865269	0.678571	0.886104	0.555195
8	XGBoost	0.919162	0.690476	0.912401	0.612676	0.919162	0.690476	0.927157	0.550633

Hyperparameter tuning for TF-IDF:

As by doing HyperParameter
tuning we can observe as now
Adaboost accuracy is Convenient.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
1	SVM	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
2	Naive Bayes	0.311377	0.166667	0.272229	0.101936	0.526854	0.116667	0.347652	0.179545
3	KNN	0.739521	0.738095	0.200304	0.171034	0.215280	0.196825	0.448036	0.151220
4	Random Forest	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
5	Bagging	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
6	AdaBoost	0.745509	0.714286	0.206874	0.166667	0.219807	0.190476	0.317587	0.148148
7	Gradient Boost	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
8	XGBoost	0.772455	0.726190	0.325300	0.168276	0.294588	0.183651	0.647421	0.148780

★ Using CBOW:

In the CBOW model, the distributed representations of context (or surrounding words) are combined to predict the word in the middle .

Logistic regression and SVM performs better.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
1	SVM	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
2	Naive Bayes	0.176647	0.071429	0.156929	0.061334	0.176647	0.071429	0.714938	0.509601
3	KNN	0.751497	0.690476	0.688169	0.612676	0.751497	0.690476	0.676154	0.550633
4	Random Forest	0.997006	0.750000	0.996996	0.642857	0.997006	0.750000	0.997097	0.562500
5	Bagging	0.970060	0.750000	0.969260	0.642857	0.970060	0.750000	0.970914	0.562500
6	AdaBoost	0.694611	0.619048	0.671947	0.582090	0.694611	0.619048	0.699500	0.549296
7	Gradient Boost	0.943114	0.726190	0.940008	0.631034	0.943114	0.726190	0.947192	0.557927
8	XGBoost	0.997006	0.738095	0.996996	0.636986	0.997006	0.738095	0.997097	0.560241

Hyperparameter tuning for CBOW:

As by doing HyperParameter tuning we can observe as all models except naive bayes ,gradient boost performs well.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
1	SVM	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
2	Naive Bayes	0.176647	0.071429	0.188834	0.055129	0.416220	0.142063	0.314889	0.159140
3	KNN	0.742515	0.738095	0.196649	0.169863	0.213657	0.196825	0.548193	0.149398
4	Random Forest	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
5	Bagging	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
6	AdaBoost	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
7	Gradient Boost	0.997006	0.750000	0.992479	0.171429	0.991304	0.200000	0.993939	0.150000
8	XGBoost	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000

★ Using Skipgram:

Skip-gram is used to predict the context word for a given target word. It's reverse

Logistic regression
and kNN, SVM performs better.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
1	SVM	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
2	Naive Bayes	0.212575	0.130952	0.203380	0.153187	0.212575	0.130952	0.726061	0.518737
3	KNN	0.730539	0.726190	0.640620	0.649676	0.730539	0.726190	0.610832	0.593430
4	Random Forest	0.997006	0.750000	0.997014	0.642857	0.997006	0.750000	0.997131	0.562500
5	Bagging	0.973054	0.750000	0.972186	0.642857	0.973054	0.750000	0.973672	0.562500
6	AdaBoost	0.592814	0.559524	0.605407	0.581132	0.592814	0.559524	0.630337	0.639776
7	Gradient Boost	0.961078	0.690476	0.959431	0.612676	0.961078	0.690476	0.962613	0.550633
8	XGBoost	0.997006	0.726190	0.997014	0.652850	0.997006	0.726190	0.997131	0.617238

Hyperparameter tuning for Word2vec-skipgram:

As by doing HyperParameter tuning we can observe as all models except naive bayes ,XG boost performs well.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.1500
1	SVM	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.1500
2	Naive Bayes	0.212575	0.130952	0.204689	0.105291	0.402106	0.179762	0.294484	0.1806
3	KNN	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.1500
4	Random Forest	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.1500
5	Bagging	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.1500
6	AdaBoost	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.1500
7	Gradient Boost	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.1500
8	XGBoost	0.907186	0.750000	0.673837	0.171429	0.614795	0.200000	0.764928	0.1500

★ Using Glove :

Logistic regression
and kNN, SVM performs better.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
1	SVM	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
2	Naive Bayes	0.742515	0.511905	0.769957	0.534360	0.742515	0.511905	0.849198	0.564018
3	KNN	0.748503	0.750000	0.660700	0.668868	0.748503	0.750000	0.744629	0.622917
4	Random Forest	0.997006	0.750000	0.997014	0.642857	0.997006	0.750000	0.997131	0.562500
5	Bagging	0.970060	0.738095	0.968900	0.636986	0.970060	0.738095	0.971011	0.560241
6	AdaBoost	0.485030	0.488095	0.518215	0.537451	0.485030	0.488095	0.588020	0.634386
7	Gradient Boost	0.982036	0.678571	0.981750	0.606383	0.982036	0.678571	0.982334	0.548077
8	XGBoost	0.997006	0.714286	0.996996	0.625000	0.997006	0.714286	0.997097	0.555556

Hyperparameter tuning for Glove:

As by doing HyperParameter
tuning we can observe as
all models except naive bayes
,Gradient boost performs well.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
1	SVM	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
2	Naive Bayes	0.742515	0.511905	0.700165	0.188011	0.818961	0.205158	0.653027	0.166833
3	KNN	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
4	Random Forest	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
5	Bagging	0.757485	0.750000	0.271565	0.171429	0.256262	0.200000	0.561703	0.150000
6	AdaBoost	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
7	Gradient Boost	0.976048	0.750000	0.964018	0.171429	0.942794	0.200000	0.987570	0.150000
8	XGBoost	0.739521	0.750000	0.184234	0.171429	0.207407	0.200000	0.347748	0.150000

★ Using Fasttext :

Logistic regression
and kNN, SVM performs better.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
1	SVM	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
2	Naive Bayes	0.104790	0.119048	0.077312	0.059712	0.104790	0.119048	0.686901	0.397433
3	KNN	0.757485	0.714286	0.677313	0.629371	0.757485	0.714286	0.711954	0.562500
4	Random Forest	0.997006	0.714286	0.997014	0.625000	0.997006	0.714286	0.997131	0.555556
5	Bagging	0.967066	0.750000	0.965417	0.642857	0.967066	0.750000	0.968099	0.562500
6	AdaBoost	0.658683	0.630952	0.637519	0.597048	0.658683	0.630952	0.643546	0.566667
7	Gradient Boost	0.919162	0.726190	0.912932	0.650535	0.919162	0.726190	0.927157	0.657738
8	XGBoost	0.985030	0.690476	0.984778	0.612676	0.985030	0.690476	0.985222	0.550633

Hyperparameter tuning for Fasttext:

As by doing HyperParameter
tuning we can observe as
Log, SVM , Bagging and Random
Forest performs well.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
1	SVM	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
2	Naive Bayes	0.095808	0.107143	0.082889	0.115052	0.265668	0.296032	0.116534	0.169231
3	KNN	0.997006	0.750000	0.992479	0.171429	0.991304	0.200000	0.993939	0.150000
4	Random Forest	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
5	Bagging	0.739521	0.738095	0.182069	0.169863	0.206250	0.196825	0.347748	0.149398
6	AdaBoost	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
7	Gradient Boost	0.937126	0.750000	0.905785	0.171429	0.854338	0.200000	0.984270	0.150000
8	XGBoost	0.781437	0.750000	0.347002	0.171429	0.305847	0.200000	0.714717	0.150000

★ Using Doc2Vec :

Logistic regression
and kNN, SVM performs better.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
1	SVM	0.736527	0.750000	0.624778	0.642857	0.736527	0.750000	0.542472	0.562500
2	Naive Bayes	0.535928	0.500000	0.573396	0.548155	0.535928	0.500000	0.658831	0.620220
3	KNN	0.739521	0.750000	0.638427	0.642857	0.739521	0.750000	0.602370	0.562500
4	Random Forest	1.000000	0.750000	1.000000	0.642857	1.000000	0.750000	1.000000	0.562500
5	Bagging	0.958084	0.726190	0.956388	0.645244	0.958084	0.726190	0.960341	0.586310
6	AdaBoost	0.667665	0.666667	0.610376	0.617914	0.667665	0.666667	0.574979	0.577679
7	Gradient Boost	0.919162	0.666667	0.913518	0.604317	0.919162	0.666667	0.927157	0.552632
8	XGBoost	1.000000	0.750000	1.000000	0.642857	1.000000	0.750000	1.000000	0.562500

Hyperparameter tuning for Doc2Vec:

As by doing HyperParameter
tuning we can observe as
All models perform well except
Naive Bayes.

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
1	SVM	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
2	Naive Bayes	0.529940	0.500000	0.386396	0.279426	0.522538	0.296429	0.366757	0.287879
3	KNN	0.739521	0.750000	0.184234	0.171429	0.207407	0.200000	0.347748	0.150000
4	Random Forest	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
5	Bagging	0.736527	0.761905	0.169655	0.252603	0.200000	0.250000	0.147305	0.351807
6	AdaBoost	0.739521	0.750000	0.181966	0.171429	0.206250	0.200000	0.215307	0.150000
7	Gradient Boost	0.736527	0.750000	0.169655	0.171429	0.200000	0.200000	0.147305	0.150000
8	XGBoost	0.739521	0.750000	0.182069	0.171429	0.206250	0.200000	0.347748	0.150000

★ DESIGN AND TRAIN MODEL USING NEURAL NETWORK:

❖ FOR BOW :

1. For First layer :-Relu Activation
2. For Second Layer : Relu Activation
3. For Last Layer : Softmax Activation

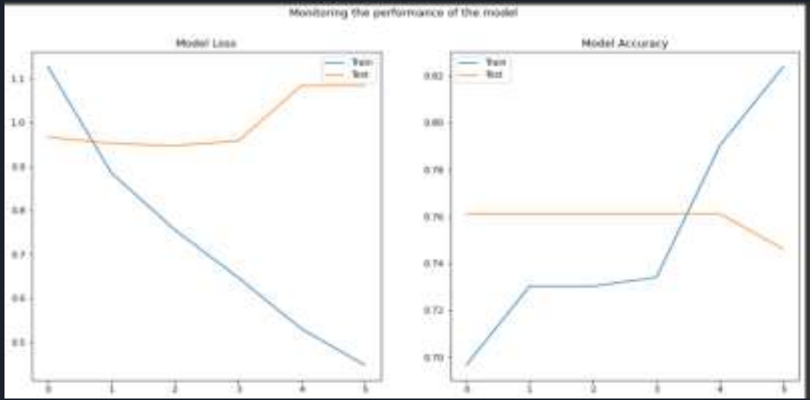
```
[ ] history=bow_model.fit(X_train_bow.toarray(), y_train_dummy, validation_split = 0.2, epochs=epochs, batch_size=batch_size, callbacks=[early_stopping])

Epoch 1/100
23/23 [==============================] - 1s 10ms/step - loss: 1.1273 - accuracy: 0.6966 - val_loss: 0.9670 - val_accuracy: 0.7612
Epoch 2/100
23/23 [==============================] - 0s 4ms/step - loss: 0.8858 - accuracy: 0.7303 - val_loss: 0.9524 - val_accuracy: 0.7612
Epoch 3/100
23/23 [==============================] - 0s 4ms/step - loss: 0.7559 - accuracy: 0.7303 - val_loss: 0.9475 - val_accuracy: 0.7612
Epoch 4/100
23/23 [==============================] - 0s 4ms/step - loss: 0.6472 - accuracy: 0.7341 - val_loss: 0.9582 - val_accuracy: 0.7612
Epoch 5/100
23/23 [==============================] - 0s 4ms/step - loss: 0.5305 - accuracy: 0.7903 - val_loss: 1.0839 - val_accuracy: 0.7612
Epoch 6/100
23/23 [==============================] - 0s 4ms/step - loss: 0.4477 - accuracy: 0.8240 - val_loss: 1.0842 - val_accuracy: 0.7463
```

```
# evaluate the keras model
_, train_accuracy = bow_model.evaluate(X_train_bow.toarray(), y_train_dummy, batch_size=8, verbose=0)
_, test_accuracy = bow_model.evaluate(X_test_bow.toarray(), y_test_dummy, batch_size=8, verbose=0)

print('Train accuracy: %.2f' % (train_accuracy*100))
print('Test accuracy: %.2f' % (test_accuracy*100))

Train accuracy: 85.63
Test accuracy: 71.43
```

Acuuracy Score

Train – 85.63%
Test – 71.43%

Model Loss



Model Accuracy

★ DESIGN AND TRAIN MODEL USING NEURAL NETWORK:

❖ FOR TF -IDF :

1. For First layer :-Relu Activation
2. For Second Layer : Relu Activation
3. For Last Layer : Softmax Activation

```
history=tfidf_model.fit(X_train_tf.toarray(), y_train_dummy, validation_split = 0.2, epochs=epochs, batch_size=batch_size, callbacks=[early_stopping])

Epoch 1/100
23/23 [==============================] - 1s 9ms/step - loss: 1.1687 - accuracy: 0.6891 - val_loss: 0.8887 - val_accuracy: 0.7612
Epoch 2/100
23/23 [==============================] - 0s 3ms/step - loss: 0.8983 - accuracy: 0.7303 - val_loss: 0.8717 - val_accuracy: 0.7612
Epoch 3/100
23/23 [==============================] - 0s 3ms/step - loss: 0.8205 - accuracy: 0.7303 - val_loss: 0.8885 - val_accuracy: 0.7612
Epoch 4/100
23/23 [==============================] - 0s 4ms/step - loss: 0.7653 - accuracy: 0.7303 - val_loss: 0.8917 - val_accuracy: 0.7612
Epoch 5/100
23/23 [==============================] - 0s 3ms/step - loss: 0.7098 - accuracy: 0.7303 - val_loss: 0.9407 - val_accuracy: 0.7612
```

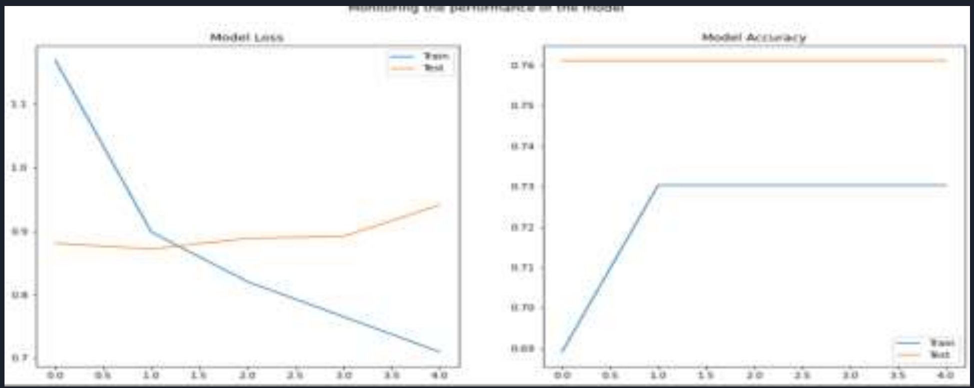
```
# evaluate the keras model
_, train_accuracy = tfidf_model.evaluate(X_train_tf.toarray(), y_train_dummy, batch_size=8, verbose=0)
_, test_accuracy = tfidf_model.evaluate(X_test_tf.toarray(), y_test_dummy, batch_size=8, verbose=0)

print('Train accuracy: %.2f' % (train_accuracy*100))
print('Test accuracy: %.2f' % (test_accuracy*100))

Train accuracy: 73.65
Test accuracy: 75.00
```

Acuuracy Score

Train – 73.65%
Test - 75%

(TF-IDF is better than BOW)

Model Loss



Model Accuracy

★ DESIGN AND TRAIN MODEL USING NEURAL NETWORK:

❖ FOR WORD TO VEC :

1. For First layer :-Relu Activation
2. For Second Layer : Relu Activation
3. For Last Layer : Softmax Activation

```
history=cbow_model.fit(xtrain_w2v, y_train_dummy, validation_data=(xtest_w2v, y_test_dummy), epochs=epochs, batch_size=batch_size, callbacks=callbacks)

Epoch 1/100
34/34 [==============================] - 1s 7ms/step - loss: 1.1388 - accuracy: 0.7096 - val_loss: 0.9075 - val_accuracy: 0.7500
Epoch 2/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9417 - accuracy: 0.7365 - val_loss: 0.9049 - val_accuracy: 0.7500
Epoch 3/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9319 - accuracy: 0.7365 - val_loss: 0.8996 - val_accuracy: 0.7500
Epoch 4/100
34/34 [==============================] - 0s 4ms/step - loss: 0.9367 - accuracy: 0.7365 - val_loss: 0.8991 - val_accuracy: 0.7500
Epoch 5/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9382 - accuracy: 0.7365 - val_loss: 0.9040 - val_accuracy: 0.7500
Epoch 6/100
34/34 [==============================] - 0s 4ms/step - loss: 0.9300 - accuracy: 0.7365 - val_loss: 0.8965 - val_accuracy: 0.7500
Epoch 7/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9267 - accuracy: 0.7365 - val_loss: 0.9090 - val_accuracy: 0.7500
Epoch 8/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9291 - accuracy: 0.7365 - val_loss: 0.9038 - val_accuracy: 0.7500
Epoch 9/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9283 - accuracy: 0.7365 - val_loss: 0.9010 - val_accuracy: 0.7500
```

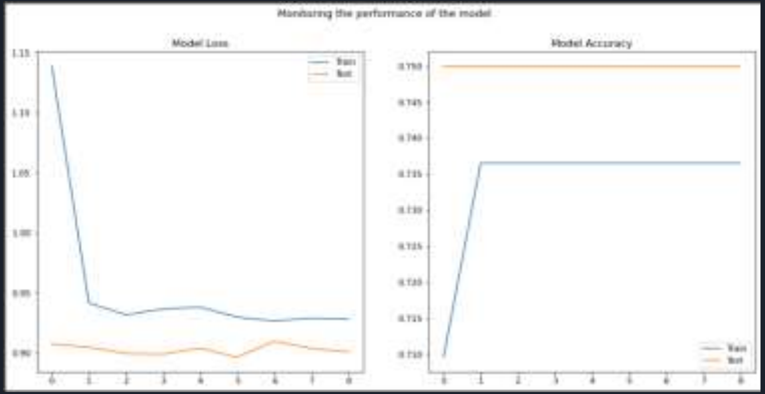
```
# evaluate the keras model
_, train_accuracy = cbow_model.evaluate(xtrain_w2v, y_train_dummy, batch_size=8, verbose=0)
_, test_accuracy = cbow_model.evaluate(xtest_w2v, y_test_dummy, batch_size=8, verbose=0)

print('Train accuracy: %.2f' % (train_accuracy*100))
print('Test accuracy: %.2f' % (test_accuracy*100))

Train accuracy: 73.65
Test accuracy: 75.00
```

Accuracy Score

Train – 73.65%
Test - 75%

(Word To VEC is similar to TF-IDF)

Model Loss



Model Accuracy

★ DESIGN AND TRAIN MODEL USING NEURAL NETWORK:

❖ FOR SKIPGRAM :

1. For First layer :-Relu Activation
2. For Second Layer : Relu Activation
3. For Last Layer : Softmax Activation

```
[ ] history=skipgram_model.fit(xtrain_w2v_sg, y_train_dummy, validation_data=(xtest_w2v_sg, y_test_dummy), epochs=epochs, batch_size=batch_size, callbacks=callbacks)

Epoch 1/100
34/34 [==============================] - 1s 8ms/step - loss: 1.0295 - accuracy: 0.7096 - val_loss: 0.9129 - val_accuracy: 0.7500
Epoch 2/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9284 - accuracy: 0.7365 - val_loss: 0.8932 - val_accuracy: 0.7500
Epoch 3/100
34/34 [==============================] - 0s 4ms/step - loss: 0.9320 - accuracy: 0.7365 - val_loss: 0.8911 - val_accuracy: 0.7500
Epoch 4/100
34/34 [==============================] - 0s 4ms/step - loss: 0.9377 - accuracy: 0.7365 - val_loss: 0.8957 - val_accuracy: 0.7500
Epoch 5/100
34/34 [==============================] - 0s 4ms/step - loss: 0.9397 - accuracy: 0.7365 - val_loss: 0.8954 - val_accuracy: 0.7500
Epoch 6/100
34/34 [==============================] - 0s 4ms/step - loss: 0.9295 - accuracy: 0.7365 - val_loss: 0.8936 - val_accuracy: 0.7500
```

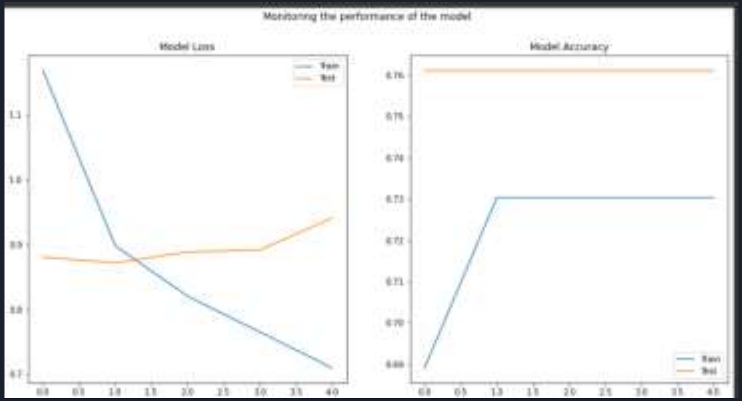
```
# evaluate the keras model
_, train_accuracy = skipgram_model.evaluate(xtrain_w2v_sg, y_train_dummy, batch_size=8, verbose=0)
_, test_accuracy = skipgram_model.evaluate(xtest_w2v_sg, y_test_dummy, batch_size=8, verbose=0)

print('Train accuracy: %.2f' % (train_accuracy*100))
print('Test accuracy: %.2f' % (test_accuracy*100))

Train accuracy: 73.65
Test accuracy: 75.00
```

Accuracy Score

Train – 73.65%
Test - 75%

(SKIPGRAM is similar to Word To VEC)

Model Loss



Model Accuracy

★ DESIGN AND TRAIN MODEL USING NEURAL NETWORK:

❖ FOR FASTTEXT :

1. For First layer :-Relu Activation
2. For Second Layer : Relu Activation
3. For Last Layer : Softmax Activation

```
history=fasttext_model.fit(xtrain_ft, y_train_dummy, validation_data=(xtest_ft, y_test_dummy), epochs=epochs, batch_size=batch_size, callbacks=callbacks)

Epoch 1/100
34/34 [==============================] - 1s 7ms/step - loss: 1.0119 - accuracy: 0.7090 - val_loss: 0.9030 - val_accuracy: 0.7500
Epoch 2/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9292 - accuracy: 0.7365 - val_loss: 0.8884 - val_accuracy: 0.7500
Epoch 3/100
34/34 [==============================] - 0s 4ms/step - loss: 0.9298 - accuracy: 0.7365 - val_loss: 0.8852 - val_accuracy: 0.7500
Epoch 4/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9371 - accuracy: 0.7365 - val_loss: 0.8908 - val_accuracy: 0.7500
Epoch 5/100
34/34 [==============================] - 0s 3ms/step - loss: 0.9367 - accuracy: 0.7365 - val_loss: 0.8870 - val_accuracy: 0.7500
Epoch 6/100
34/34 [==============================] - 0s 4ms/step - loss: 0.9264 - accuracy: 0.7365 - val_loss: 0.8882 - val_accuracy: 0.7500
```

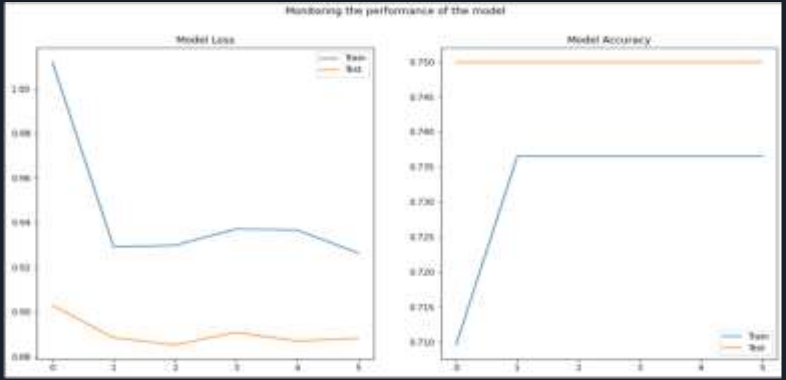
```
# evaluate the keras model
_, train_accuracy = fasttext_model.evaluate(xtrain_ft, y_train_dummy, batch_size=8, verbose=0)
_, test_accuracy = fasttext_model.evaluate(xtest_ft, y_test_dummy, batch_size=8, verbose=0)

print('Train accuracy: %.2f' % (train_accuracy*100))
print('Test accuracy: %.2f' % (test_accuracy*100))

Train accuracy: 73.65
Test accuracy: 75.00
```

Accuracy Score

Train – 73.65%
Test - 75%

(FASTTEXT is similar to SKIPGRAM)

Model Loss



Model Accuracy

★ DESIGN AND TRAIN MODEL USING NEURAL NETWORK:

❖ FOR DOC2VEC :

1. For First layer :-Relu Activation
2. For Second Layer : Relu Activation
3. For Last Layer : Softmax Activation

```
history=doc2vec_model.fit(xtrain_dc, y_train_dummy, validation_data=(xtest_dc, y_test_dummy), epochs=epochs, batch_size=batch_size, callbacks=callbacks)

Epoch 1/100
34/34 [==============================] - 1s 8ms/step - loss: 1.0614 - accuracy: 0.7126 - val_loss: 0.9215 - val_accuracy: 0.7500
Epoch 2/100
34/34 [==============================] - 0s 4ms/step - loss: 0.9139 - accuracy: 0.7365 - val_loss: 0.8923 - val_accuracy: 0.7500
Epoch 3/100
34/34 [==============================] - 0s 4ms/step - loss: 0.8780 - accuracy: 0.7365 - val_loss: 0.9103 - val_accuracy: 0.7500
Epoch 4/100
34/34 [==============================] - 0s 4ms/step - loss: 0.8436 - accuracy: 0.7365 - val_loss: 0.9520 - val_accuracy: 0.7500
Epoch 5/100
34/34 [==============================] - 0s 4ms/step - loss: 0.7827 - accuracy: 0.7425 - val_loss: 0.9401 - val_accuracy: 0.7500
```

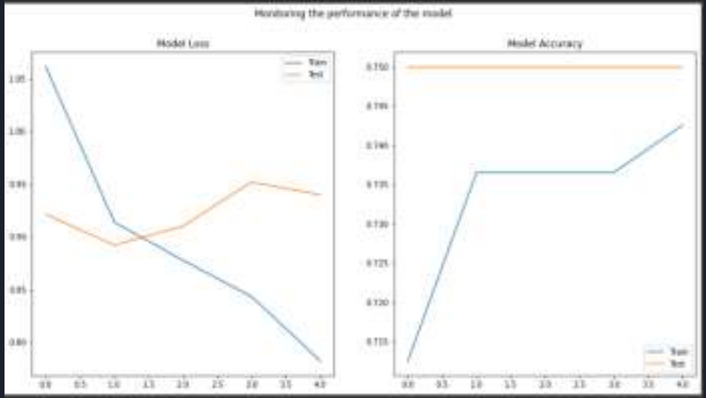
```
# evaluate the keras model
_, train_accuracy = doc2vec_model.evaluate(xtrain_dc, y_train_dummy, batch_size=8, verbose=0)
_, test_accuracy = doc2vec_model.evaluate(xtest_dc, y_test_dummy, batch_size=8, verbose=0)

print('Train accuracy: %.2f' % (train_accuracy*100))
print('Test accuracy: %.2f' % (test_accuracy*100))

Train accuracy: 73.65
Test accuracy: 75.00
```

Accuracy Score

Train – 73.65%
Test - 75%

(DoC2VEC is similar to FASTTEXT)

Model Loss



Model Accuracy

★ DESIGN AND TRAIN MODEL USING RNN LSTM CLASSIFIER:

❖ FOR GLOVE :

1. For First layer :-Embedding Layer
2. For Second layer : Bidirectional LSTM
3. For Third layer : Global Max Pool
4. For Fourth layer: Dropout

```
Epoch 97/100
42/42 [==============================] - 9s 216ms/step - loss: 0.9069 - acc: 0.7395 - val_loss: 0.9146 - val_acc: 0.7381 - lr: 1.0000e-23
Epoch 98/100
42/42 [==============================] - 9s 214ms/step - loss: 0.9078 - acc: 0.7395 - val_loss: 0.9146 - val_acc: 0.7381 - lr: 1.0000e-27
Epoch 99/100
42/42 [==============================] - 9s 215ms/step - loss: 0.9052 - acc: 0.7395 - val_loss: 0.9146 - val_acc: 0.7381 - lr: 1.0000e-27
Epoch 100/100
42/42 [==============================] - 9s 213ms/step - loss: 0.9052 - acc: 0.7395 - val_loss: 0.9146 - val_acc: 0.7381 - lr: 1.0000e-27
```

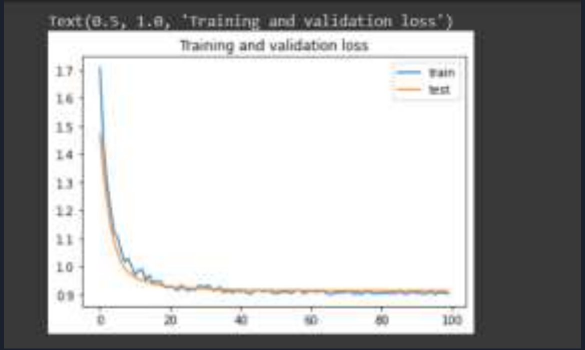
```
# evaluate the keras model
_, train_accuracy = model.evaluate(X_text_train, y_text_train, batch_size=8, verbose=0)
_, test_accuracy = model.evaluate(X_text_test, y_text_test, batch_size=8, verbose=0)

print('Train accuracy: %.2f' % (train_accuracy*100))
print('Test accuracy: %.2f' % (test_accuracy*100))
```

```
Train accuracy: 73.95
Test accuracy: 73.81
```

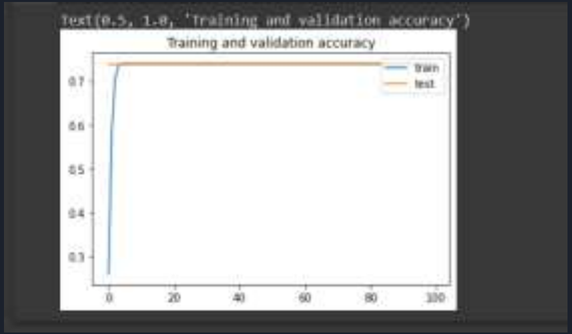
Accuracy Score

Train – 73.95%
Test – 73.81%

(GLOVE is similar to TF-IDF,
Word To VEC, SKIPGRAM, FASTTEXT
& DoC2VEC)



Model Loss



Model Accuracy

★ FIXING CLASS IMBALANCE :

❖ USING SMOTE :

X_train shape- 1158 , 93
X_test shape- 387, 93

```python
from imblearn.over_sampling import SMOTE
from collections import Counter

counter = Counter(df['Accident_Level'].to_list())
print('Before',counter)

# oversampling the train dataset using SMOTE
smt = SMOTE()
labels = df['Accident_Level'].tolist()
X_smote, y_smote = smt.fit_resample(X, y)

X_train_smote, X_test_smote, y_train_smote, y_test_smote = train_test_split(X_smote, y_smote, stratify=y_smote, random_state=1)

counter = Counter(y_train_smote)
print('After',counter)
```

```
Before Counter({0: 309, 1: 40, 2: 31, 3: 30, 4: 8})
After Counter({1: 232, 2: 232, 4: 232, 3: 231, 0: 231})
```

Accuracy Score OF ML Models After USing SMOTE.

```python
build_model_train(X_train_smote, y_train_smote, X_test_smote, y_test_smote)
```

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.849741	0.806202	0.842372	0.781922	0.849741	0.806202	0.847312	0.799626
1	SVM	0.968048	0.901809	0.967576	0.897885	0.968048	0.901809	0.968267	0.902904
2	Naive Bayes	0.365285	0.356589	0.322730	0.307809	0.365285	0.356589	0.707532	0.686898
3	KNN	0.876511	0.842377	0.855129	0.810251	0.876611	0.842377	0.895862	0.870510
4	Random Forest	1.000000	0.961240	1.000000	0.960988	1.000000	0.961240	1.000000	0.961362
5	Bagging	1.000000	0.940568	1.000000	0.938601	1.000000	0.940568	1.000000	0.941986
6	AdaBoost	0.531088	0.503876	0.530029	0.517748	0.531088	0.503876	0.577653	0.588997
7	Gradient Boost	0.964594	0.870801	0.964017	0.863954	0.964594	0.870801	0.966073	0.870032
8	XGBoost	0.994819	0.906977	0.994803	0.903848	0.994819	0.906977	0.994833	0.910956

★ TEXT AUGMENTATION :Data Augmentation (DA) Technique is a process that enables us to artificially increase training data size by generating different versions of real datasets without actually collecting the data. The data needs to be changed to preserve the class categories for better performance in the classification task.

❖ Generating more data for minority classes in dependent variable Using Back Translation :

```
df_text['Description']=df_text['Description'].apply(lambda x: translator.translate(x,lang_tgt ='pt-br'))
```

```
df_text['Description']=df_text['Description'].apply(lambda x: translator.translate(x,lang_tgt ='en'))
```

```
# Concatenating the new datapoints extracted from the backtranslation technique with the original dataset
new_df_text = pd.concat([df_text, df.drop_duplicates()]).reset_index(drop = True)
new_df_text = new_df_text.drop_duplicates()
new_df_text.count()
```

```
Date                        836
Countries                   836
Local                       836
Industry Sector             836
Accident Level              836
Potential Accident Level    836
Gender                      836
Employee or Third Party     836
Critical Risk               836
Description                 836
```

★ FIXING CLASS IMBALANCE :

❖ USING SMOTE : OVER SAMPLING

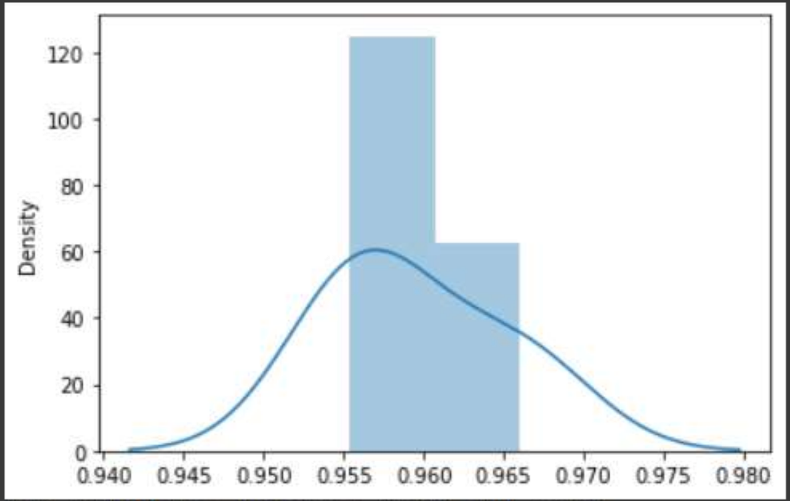
```
build_model_train(X_train_smote, y_train_smote, X_test_smote, y_test_smote)
```

	Model	Accuracy score Train	Accuracy score Test	Train F1 Score	Test F1 Score	Train Recall	Test Recall	Train Precision	Test Precision
0	LogReg	0.761761	0.738680	0.751536	0.715435	0.761761	0.738680	0.750774	0.718506
1	SVM	0.953820	0.931436	0.952886	0.929901	0.953820	0.931436	0.956564	0.934499
2	Naive Bayes	0.311178	0.322122	0.251963	0.269848	0.311178	0.322122	0.707646	0.720732
3	KNN	0.891670	0.869340	0.877572	0.848278	0.891670	0.869340	0.905280	0.890261
4	Random Forest	0.998705	0.963777	0.998706	0.963316	0.998705	0.963777	0.998714	0.964696
5	Bagging	0.998274	0.941785	0.998275	0.940269	0.998274	0.941785	0.998279	0.943446
6	AdaBoost	0.515753	0.469599	0.514916	0.470235	0.515753	0.469599	0.534918	0.485482
7	Gradient Boost	0.906776	0.844761	0.903784	0.836094	0.906776	0.844761	0.910557	0.847583
8	XGBoost	0.974968	0.891332	0.974835	0.888077	0.974968	0.891332	0.975814	0.894947

Accuracy Score OF ML Models
After USing OVER SAMPLING.

Accuracy has been improved by using SMOTE

★ CROSS VALIDATION FOR BAGGING CLASSIFIER :-




```
95% confidence interval 95.0% and 96.9%
```

```
Bagging classifier cv score
*************************
cv-mean score : 95.95%
cv-std  : 0.46%
```

★ Bagging classifier using SMOTE

Bagging classifier giving best results using smote

```
[ ] Bag_clf = BaggingClassifier(base_estimator = RandomForestClassifier(n_estimators=100, max_features='sqrt'),
                                n_estimators=200,
                                random_state=52)

    Bag_clf.fit(X_train_smote, y_train_smote) # fitting the model
    print('\nTest score:', '{:.2f}%'.format(Bag_clf.score(X_test_smote, y_test_smote)*100)) # evaluating it on test set
```

```
Test score: 96.38%
```

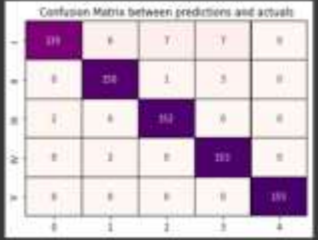
Accuracy Score - 96%

CLASSIFICATION REPORT

```
plt.title('Confusion Matrix between predictions and actuals')
sns.heatmap(cm, annot=True, fmt = '', cbar=False, cmap='RdPu', linecolor='black', linewidths=0.1);
```

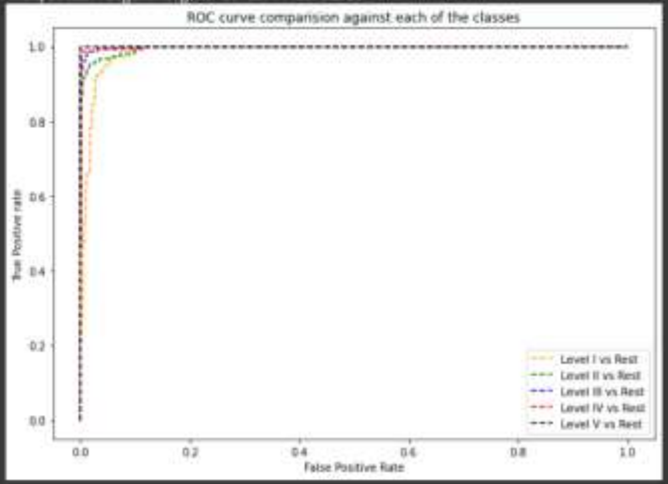
***********Classification Report***********

	precision	recall	f1-score	support
0	0.87	0.99	0.92	137
1	0.97	0.95	0.96	158
2	0.99	0.95	0.97	160
3	0.99	0.94	0.96	163
4	1.00	1.00	1.00	155
accuracy			0.96	773
macro avg	0.96	0.96	0.96	773
weighted avg	0.97	0.96	0.96	773



AUC-ROC Curve

```
<matplotlib.legend.Legend at 0x7f44d6f31510>
```



★ CHOOSE BEST PERFORMING MODEL CLASSIFIER

Bagging Classifier is the best model .
Bagging Classifier with SMOTE is best.
The Accuracy is best of this .

Step 5: Choose the best performing model classifier and pickle it.

```
#Pickle file used to store the model

import pickle
#Bag_clf.fit(X_smote, y_smote)
pickle.dump(Bag_clf, open('nlp_chatbot.sav','wb'))
```

PREDICTIONS FOR DIFFERENT ACCIDENT LEVELS

ACCIDENT LEVEL 1

For Prediction of Accident Level 1

```
[ ] predict("The worker Manuel was making the disconnection of the power cables of the gate that is at the intersection of Manco streets with Cajamarquilla in order to remove it. In circ

worker manuel making disconnection power cable gate intersection manco street cajamarquilla order remove it. circumstance mr. josa worker company its, removing rope tied body gate, yic
[0]
'Accident Level I'
```

ACCIDENT LEVEL 2

For Prediction of Accident Level 2

```
[ ] predict("During the withdrawal of the metal form support screw in the Inside of well 2, when the bolt of the chain holder was loosened, the employee and a helper exerted force on the

withdrawal metal form support screw inside well bolt chain holder loosened, employee helper exerted force combination wrench, bolt came loosen immediately, pressing ring finger employe
[1]
'Accident Level II'
```

ACCIDENT LEVEL 3

For Prediction of Accident Level 3

```
[ ] predict('In the city of Conchucos, of Ancash, participating in a patronal feast, representing the company, was mounted on a horse as part of the ceremony throwing fruits and toys to t

    city conchucos, ancash, participating patronal feast, representing company, mounted horse part ceremony throwing fruit toy people attending public event, noise material pyrotechnic pe
    [2]
    'Accident Level III'
```

ACCIDENT LEVEL 4

For Prediction of Accident Level 4

```
[ ] predict('When observing the pulp overflow of the overflow reception drawer of the thickener, the filter operator approaches to verify the operation of the C7-20 pump, making sure that

    observing pulp overflow overflow reception drawer thickener, filter operator approach verify operation pump, making sure stopped. press keypad start pump getting start, proceeds remove
    [3]
    'Accident Level IV'
```

ACCIDENT LEVEL 5

For Prediction of Accident Level 5

```
[ ] predict('Being approximately 20:57 hours of 03/22/2017; during the change of cables between the Z-332 power cell (locked cabinet) and the 2014 transformer; there is a loud noise follo

    approximately hour //; change cable power cell locked cabinet transformer; loud noise followed oscillation electrical system. moment collaborator queneche company eissa found floor hea
    [4]
    'Accident Level V'
```

UI INTERFACES OF CHATBOT

Linking Model to UI

We have used two approaches for linking our model to the UI.

1) Created a RestFul Service and hosted using Flask and consumed it using Javascript.
2) Created a UI using Tkinter Python framework.

FLASK

Flask is a **web development framework** developed in Python.

Flask is a *micro-framework* because it is *lightweight* and *only* provides components that are *essential*. It only provides the necessary components for web development, such as **routing, request handling, sessions etc.**

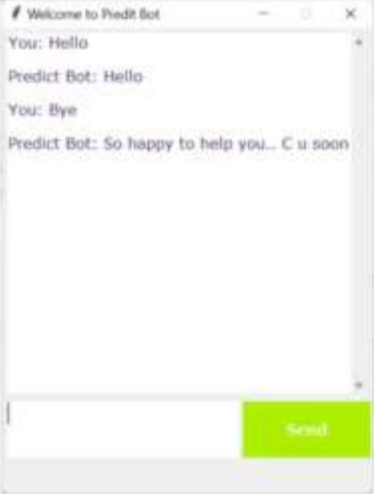
Flask Restful is an extension for Flask that adds support for building REST APIs in Python using Flask as the back-end.

Code snippet

```
@app.route("/result",methods=["POST"])
def result():
  output=request.get_json()
  if output is None:
    return {'predict':'none'}
  else:
    if len(output) != 1:
      return {"predict":"Should pass input"}
    else:
```

create Graphical User interfaces (GUIs) and is included in all standard Python Distributions. In fact, it's the only framework built into the Python standard library.

This Python framework provides an interface to the Tk toolkit and works as a thin object-oriented layer on top of Tk. The Tk toolkit is a cross-platform collection of 'graphical control elements', for building application interfaces.




```
desc=output["description"]
resp={}
res=NPro2.chatbot_response(desc)
resp["predict"]=res
return resp
```

Web UI created using HTML, CSS and service consumed using Javascript