Automatic Ticket Assignment
Capstone Project
Interim Report

Post Graduate Program in Artificial Intelligence and Machine Learning (2021-2022)

Capstone Project Title	AUTOMATIC TICKET ASSIGNMENT
Description	Apply AI techniques to classify incidents to right functional groups and help organizations reduce the resolving time and switch focus on more productive tasks.

Group 10 NLP 1 Members

Mentor	**Mr Sandeep Raghuwanshi**
Name	**Email ID**
Ankit Jain	ankit2704smit@gmail.com
Sowmiya S	ssowmya137@gmail.com
Seema Kumari Singh	seema.singh71@gmail.com
Pushpendra	spushpendra14@gmail.com
Vinoth Prakash	prakashvinith16@yahoo.com

Index

Summary of the problem statement, Data and findings

Understanding the Business

IT leverages Incident Management process to ensure there is no disruption to business operations. Any unplanned disruption can cause interruption to business services. Incident management process helps in identification of issues or problems faced by users or operation teams. Manual assignment of incidents is time consuming and requires human intervention. There may be lag in incident resolution due to human errors or if incidents are not routed appropriately. On the other hand, manual assignment also increases the response and resolution times which result in user satisfaction deterioration / poor customer service.

Background and Objective

To apply techniques and learnings to make ticket assignment more cost-effective, less resolution time so that service desk team can focus on other productive tasks. We are able to see that the current system is capable of assigning 70+% of the tickets correctly. Our target is to automatically classify tickets and directing them to appropriate groups at the earliest, helps in improving the throughput in the ticketing pipeline of an organization.

Data & Findings

Data format	CSV
Total Records	8500

Data Fields

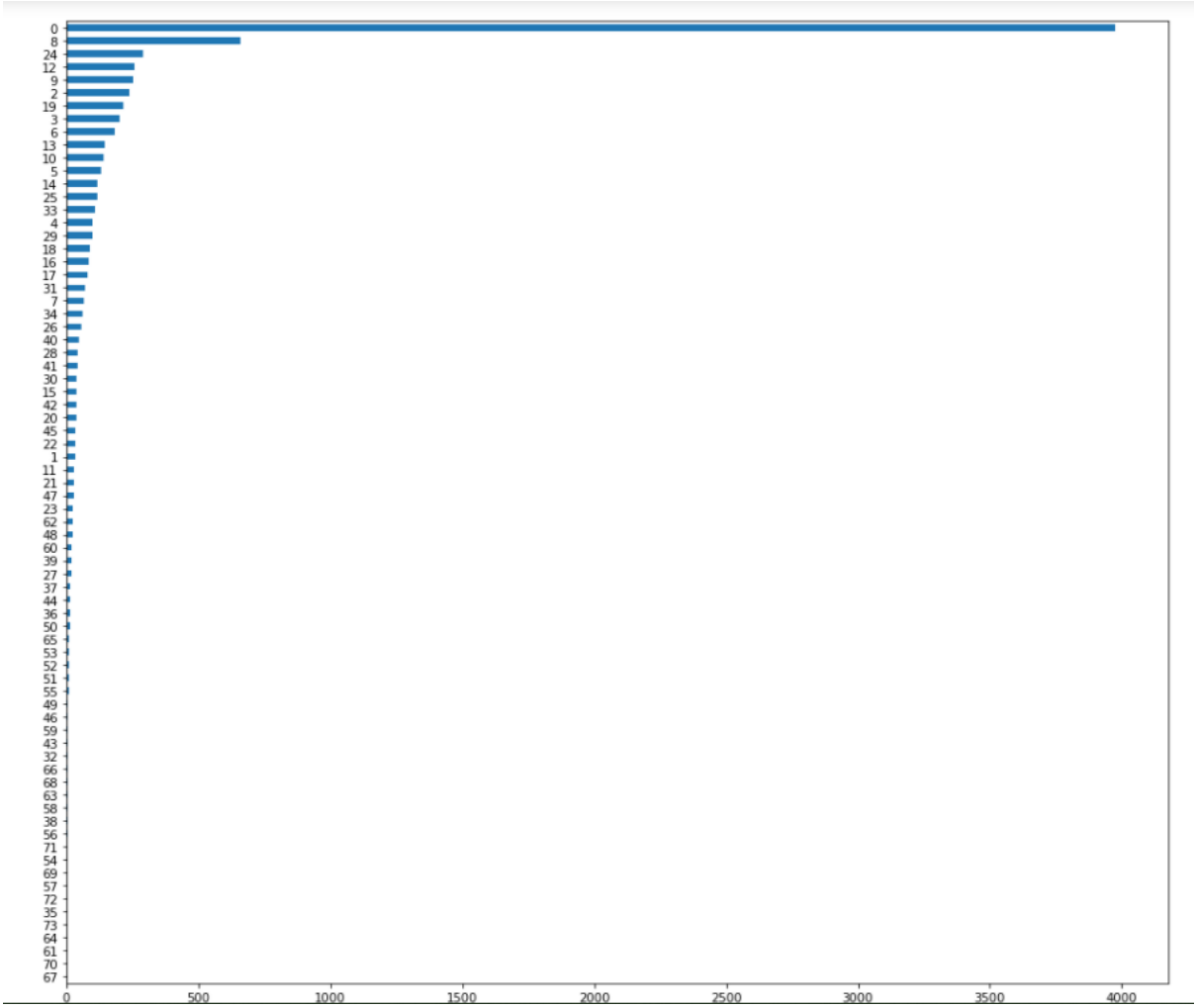
Short description	A brief overview of the issue faced by the user
Description	Detailed description of the issue
Assignment group	GRP_0 ~ GRP_73 (total 74 classes of Assignment group)

Sample data

Short description	**Description**	**Assignment group**
login issue	-verified user details.(employee# & manager name) -checked the user name in ad and reset the password. -advised the user to login and check. -caller confirmed that he was able to login. -issue resolved.	GRP_0
Outlook	received from: hmjdrvpb.komuaywn@gmail.com hello team, my meetings/skype meetings etc are not appearing in my outlook calendar, can somebody please advise how to correct this? kind	GRP_0
cant log in to vpn	received from: eylqgodm.ybqkwiam@gmail.com hi i cannot log on to vpn best	GRP_0

Observations

1. High imbalance seen in data with GRP_0 having 40%+ percent of representation and other groups with minimal records



2. Many groups/classes are with very little representation.
3. Null values: Null records are inferred after merging description and short description
4. Very few tickets have non-English descriptions
5. Four columns – Short Description, Description, Caller and Assignment group
6. 74 Assignment groups found - Target classes
7. Caller names in a random fashion (may not be useful for training data)
8. European non-English language also found in the data
9. Email/chat format in description
10. Symbols & other characters in the description
11. Hyperlinks, URLS & few image data found in the description
12. Blanks found either in the short description or description field
13. Few descriptions same as the short description
14. Few words were combined together
15. Spelling mistakes and typo errors are found

Summary of the approach to EDA and Pre-Processing

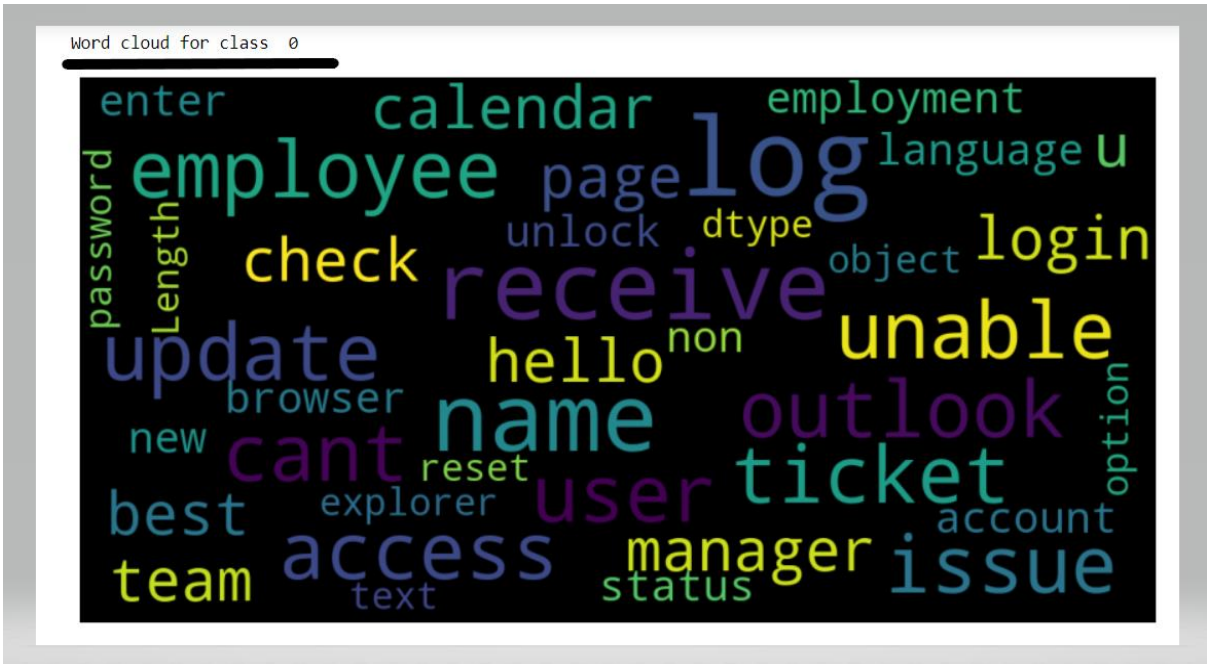
Data Pre-Processing and Cleaning

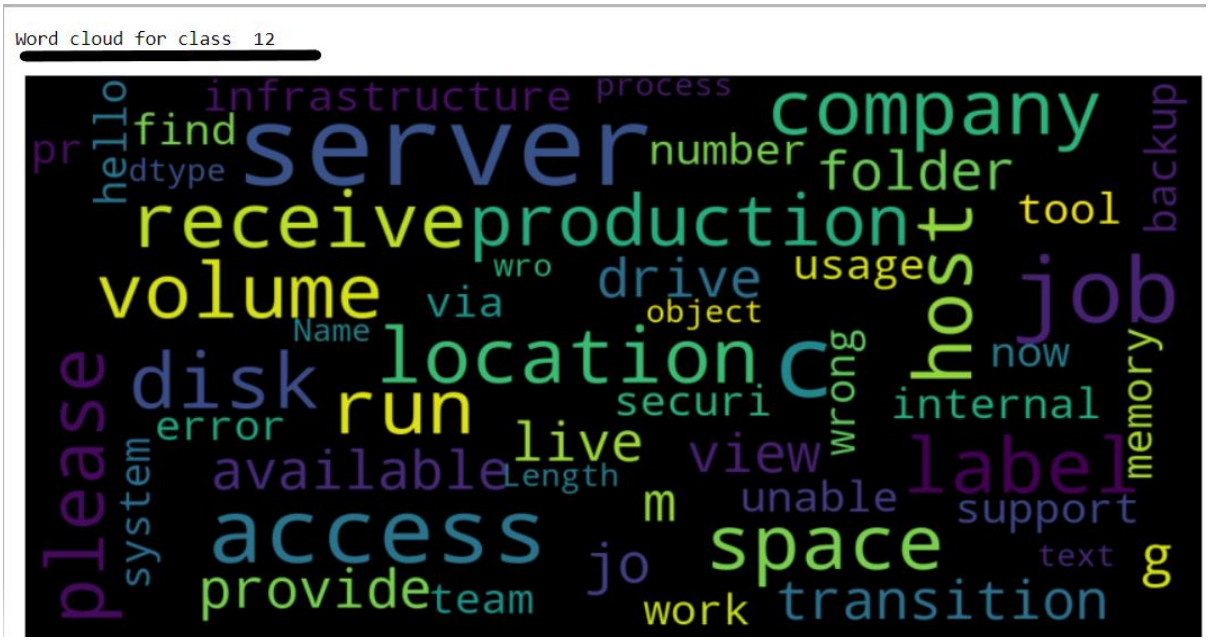
1. Dropped the caller field as the data was not found to be useful for analysis
2. Replaced Null values in short description & description with space.
3. Merged Short Description & Description fields for analysis
4. Contraction words found in the merged Description are removed for ease of word modelling
5. Changed the case sensitivity of words to the common one
6. Removed Hashtags and kept the words, Hyperlinks, URLs, HTML tags & non-ASCII symbols from merged fields.
7. Translating all languages (German) to English
8. Tokenization of merged data
9. Removal of Stop words
   English words which does not add much meaning to a sentence.
10. Lemmatization
11. WordCloud created
12. Attempted to do spell check
13. Created Plot to understand the distribution of words
14. Removal of line breaks and tabs (\r\n\t)
15. Removal of special characters
16. Removal of extra spaces

Word Cloud Distribution before Preprocessing

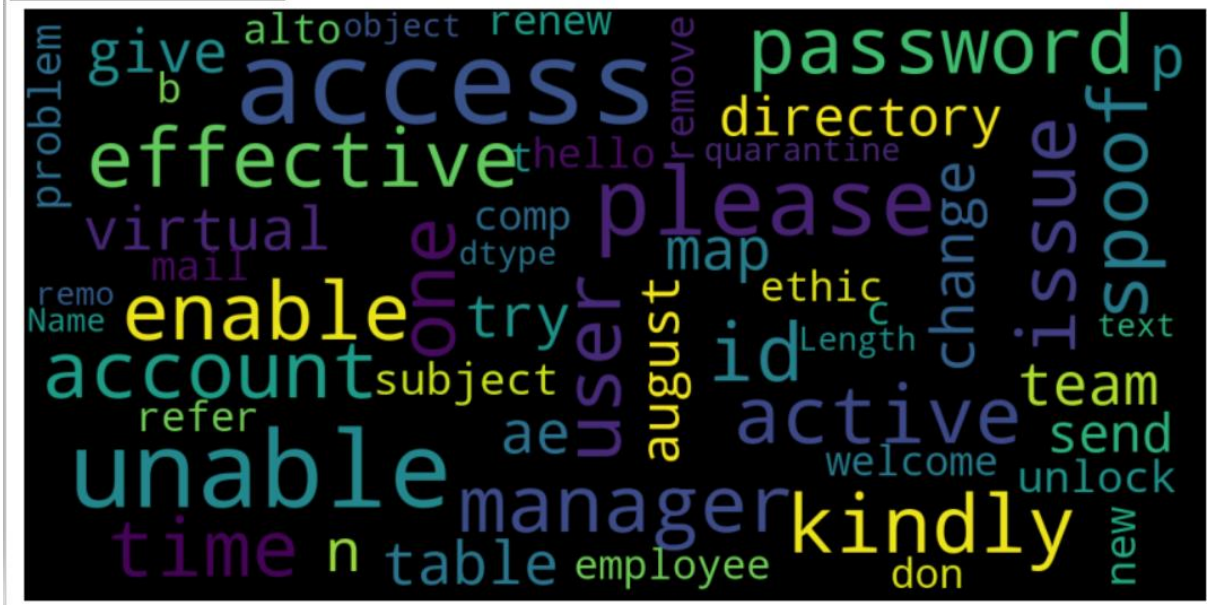
- Inclusion of Meaningless words are inferred

Word Cloud Distribution of Top 3 classes After Preprocessing

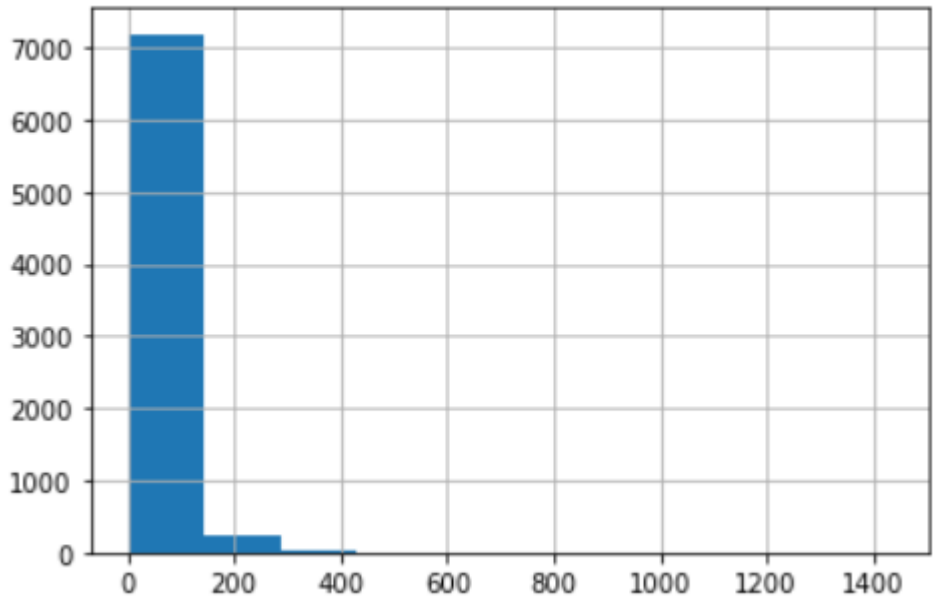




Word cloud for class 2

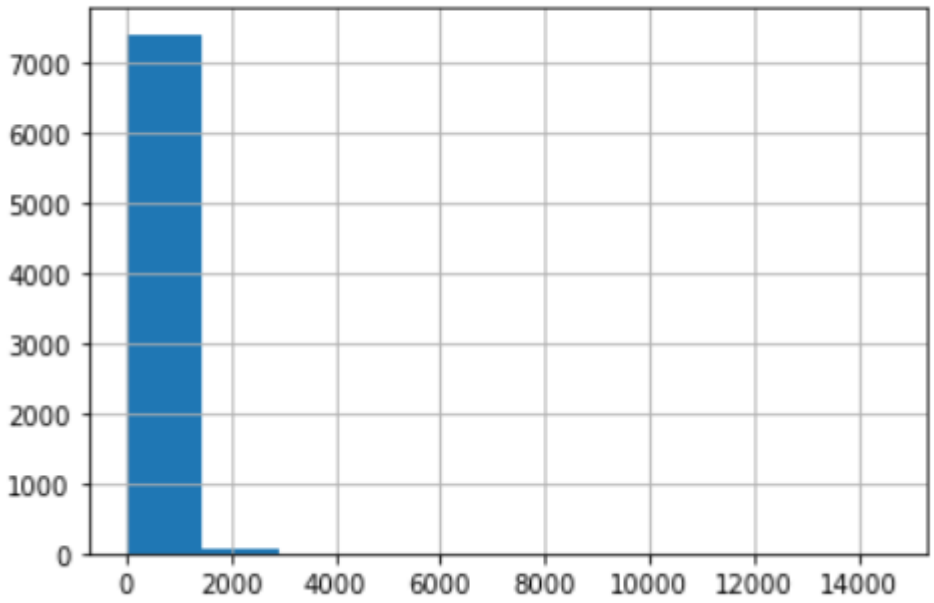


Word Length Distribution

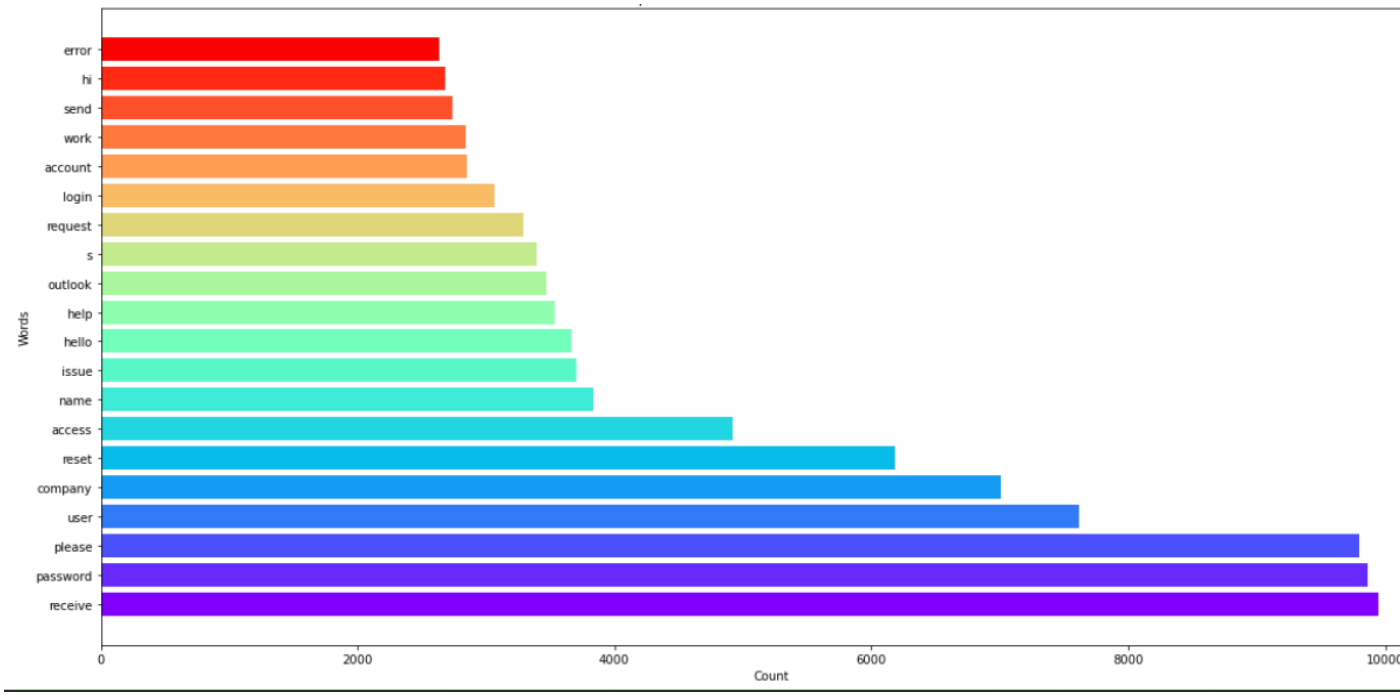


Character Length Distribution

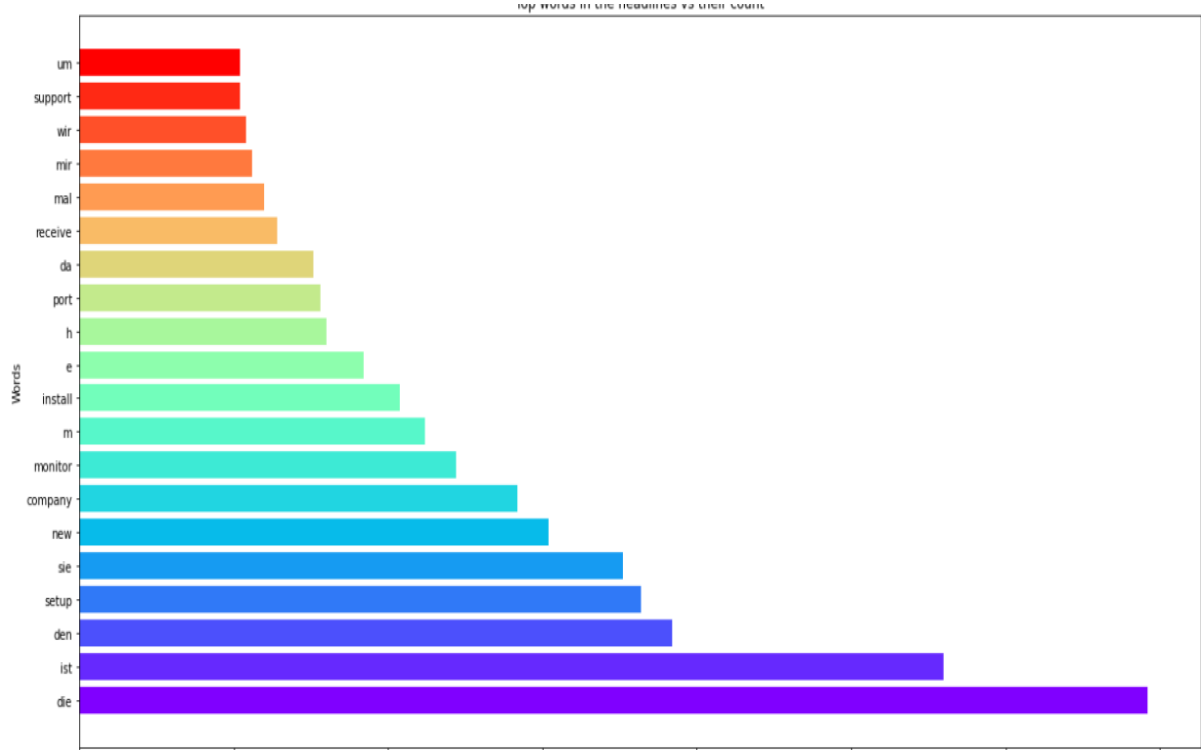
```
<matplotlib.axes._subplots.AxesSubplot at 0x22b390e5a60>
```



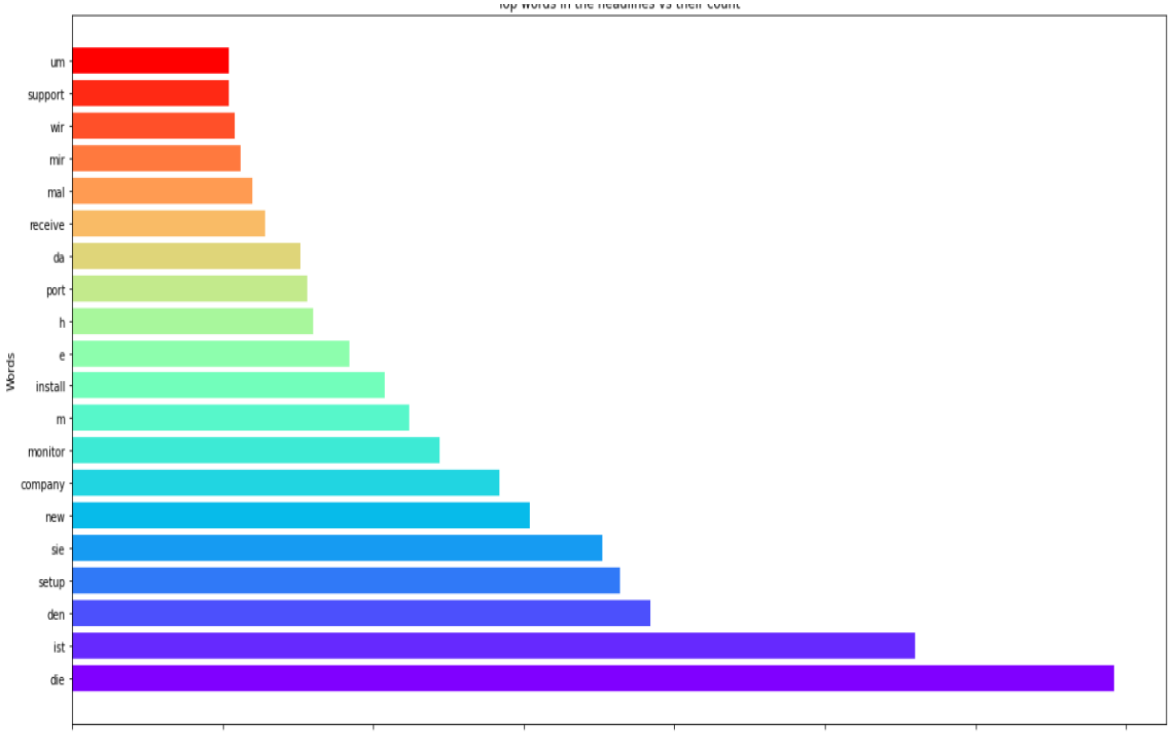
Frequently Used Words in Group 0



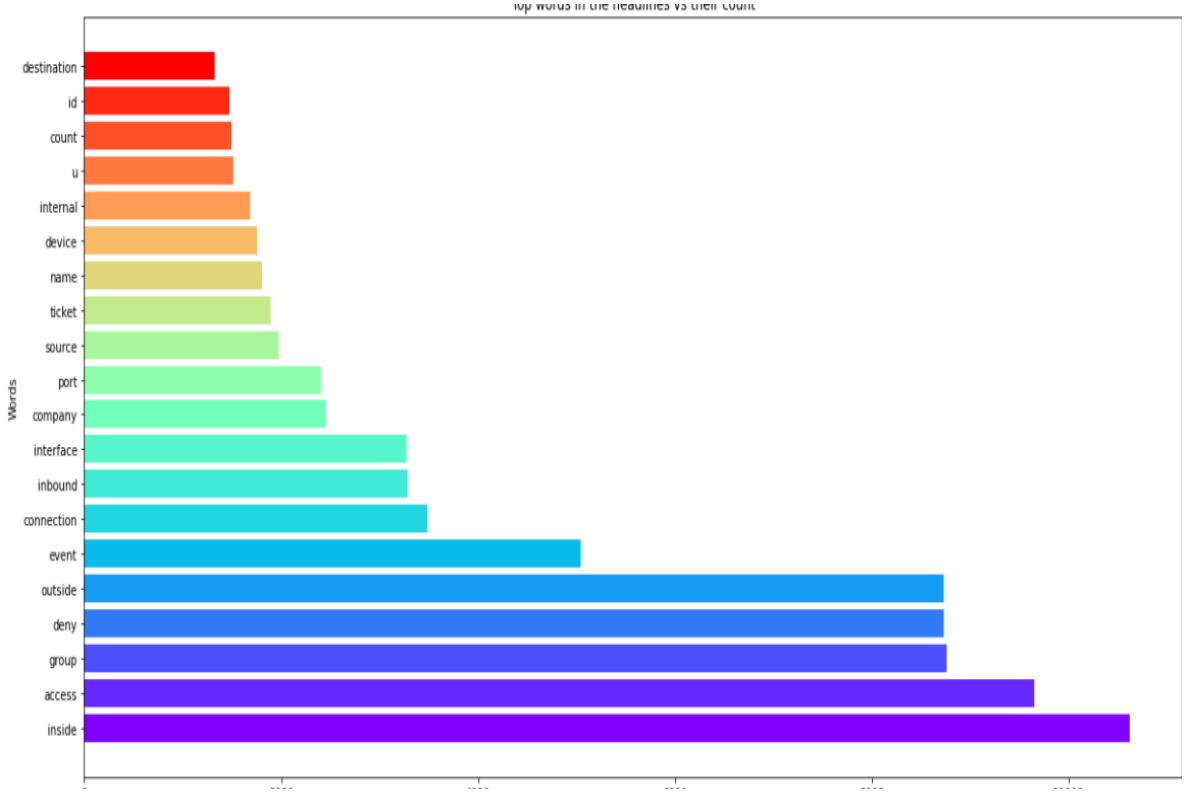
Frequently Used Words in Group 8



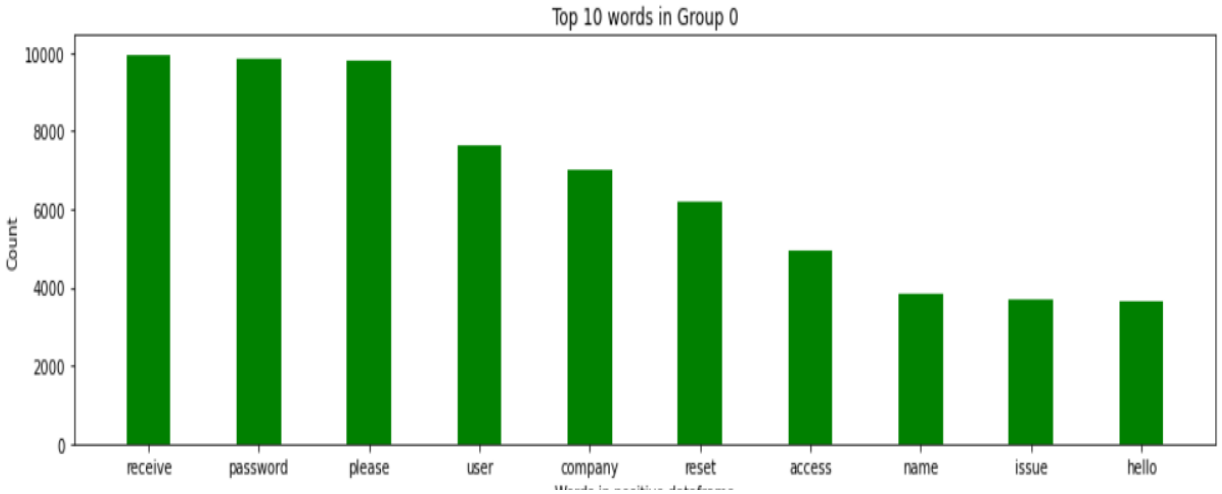
Frequently Used Words in Group 24



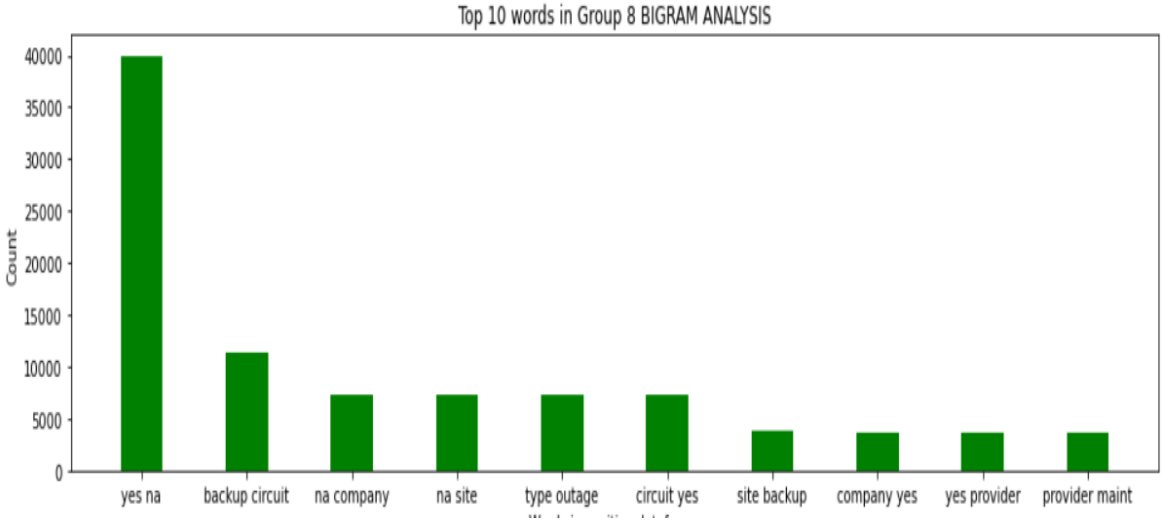
Frequently Used Words in Group 12

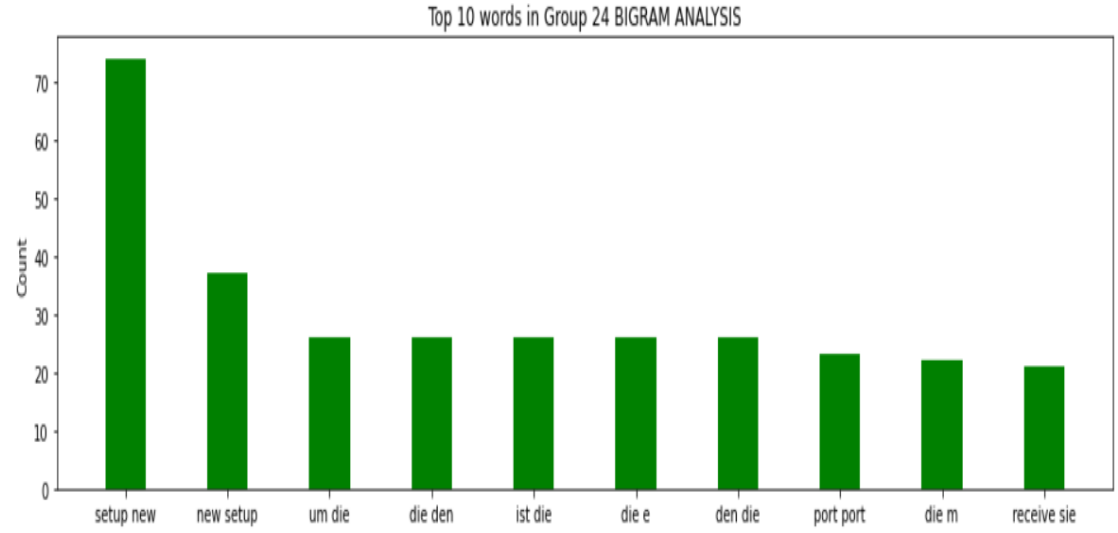


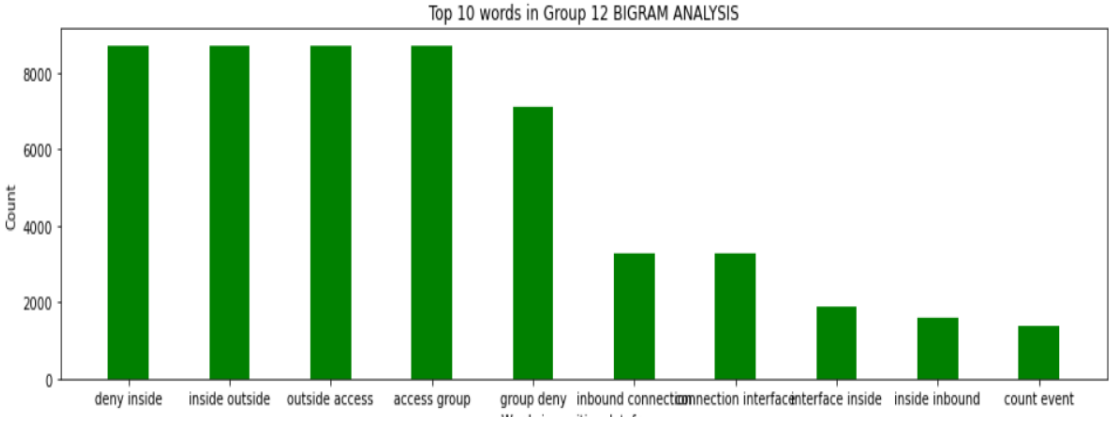
UNIGRAMS



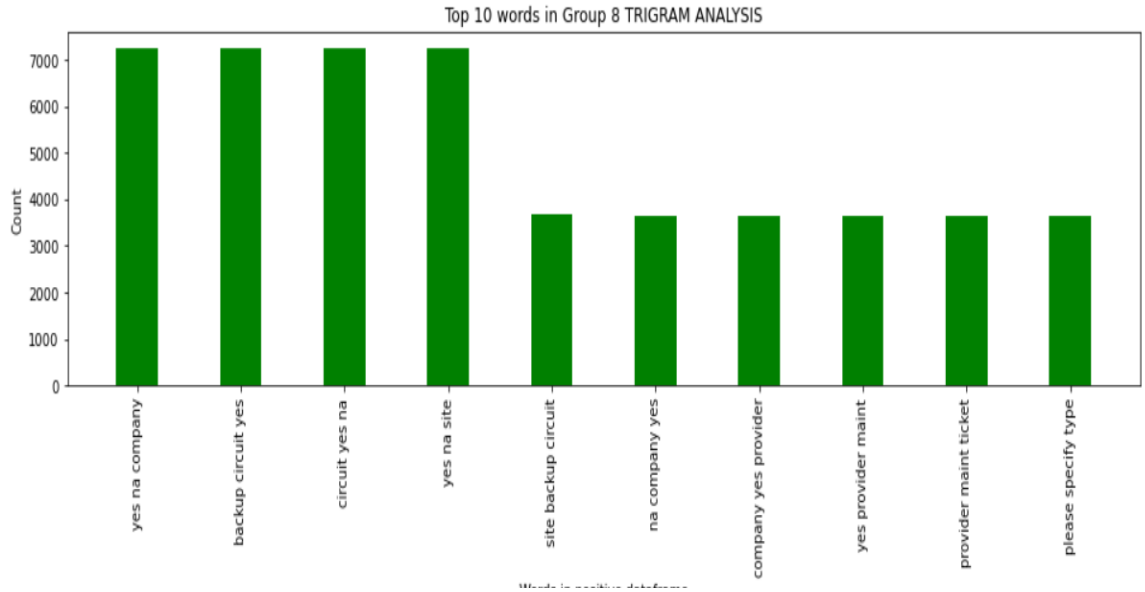
BIGRAMS

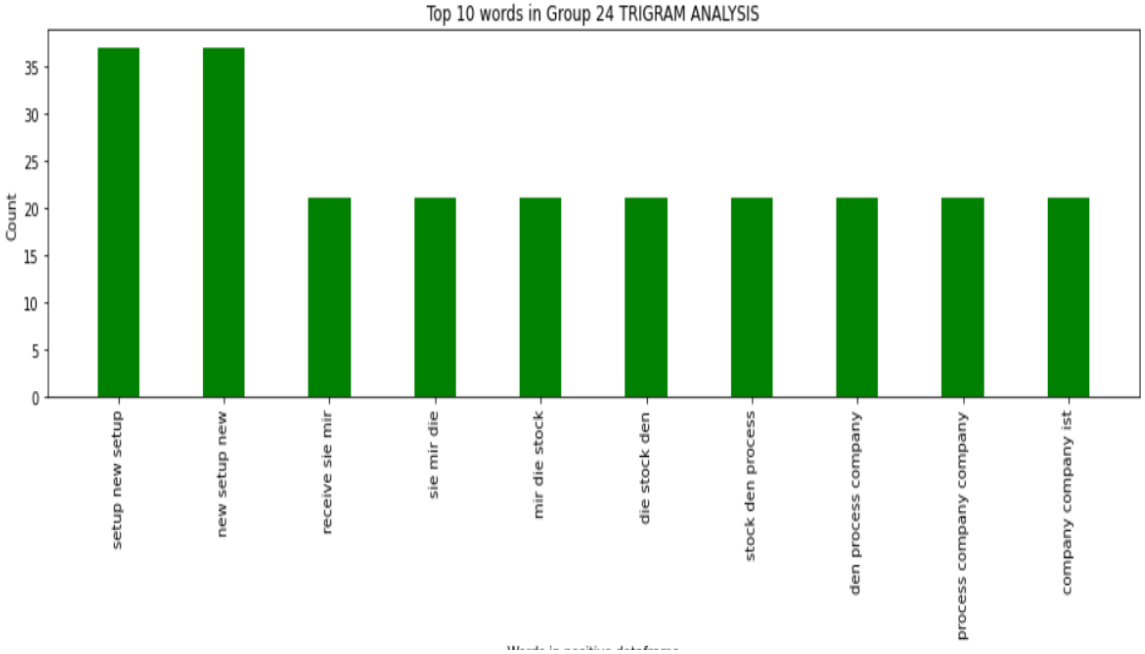

Top 10 words in Group 8 BIGRAM ANALYSIS
Count
40000
35000
30000
25000
20000
15000
10000
5000
0
yes na
backup circuit
na company
na site
type outage
circuit yes
site backup
company yes
yes provider
provider maint

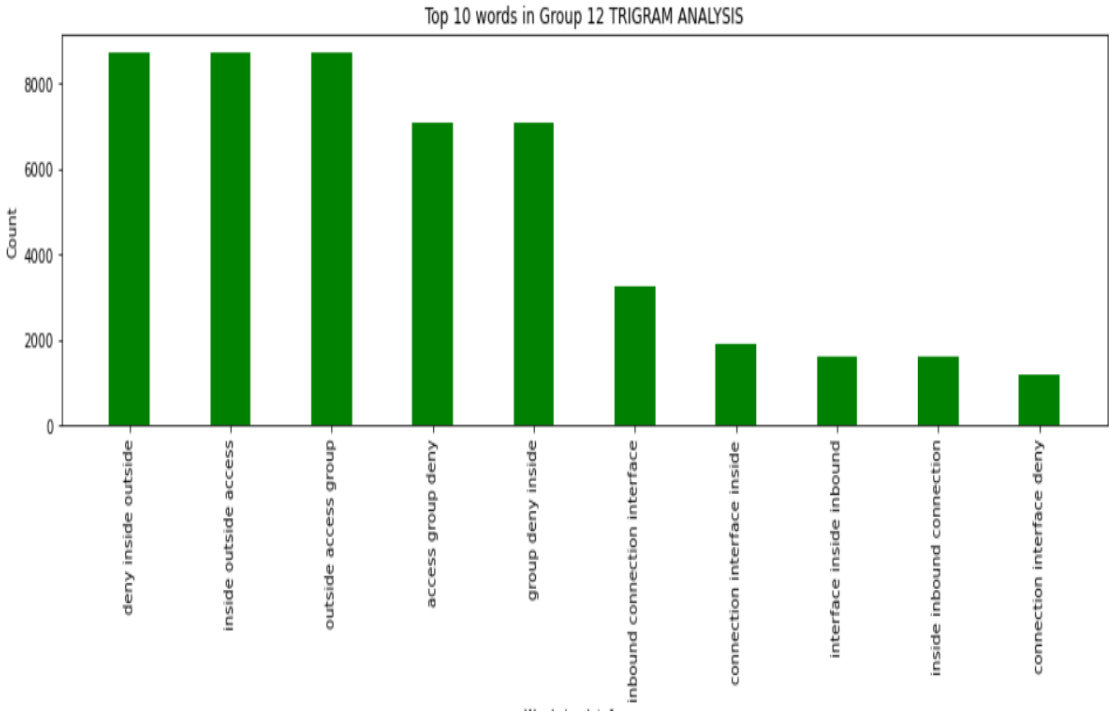

Top 10 words in Group 24 BIGRAM ANALYSIS
Count
70
60
50
40
30
20
10
0
setup new
new setup
um die
die den
ist die
die e
den die
port port
die m
receive sie


Top 10 words in Group 12 BIGRAM ANALYSIS
Count
8000
6000
4000
2000
0
deny inside
inside outside
outside access
access group
group deny
inbound connection
connection interface
interface inside
inside inbound
count event

TRIGRAMS


Top 10 words in Group 8 TRIGRAM ANALYSIS
Count
0
1000
2000
3000
4000
5000
6000
7000
yes na company
backup circuit yes
circuit yes na
yes na site
site backup circuit
na company yes
company yes provider
yes provider maint
provider maint ticket
please specify type


Top 10 words in Group 24 TRIGRAM ANALYSIS
Count
0
5
10
15
20
25
30
35
setup new setup
new setup new
receive sie mir
sie mir die
mir die stock
die stock den
stock den process
den process company
process company company
company company ist



Unigrams or 1-grams

To generate 1-grams we pass the value of n=1 in ngrams function of NLTK. But first, we split the sentence into tokens and then pass these tokens to ngrams function.

Bigrams or 2-grams

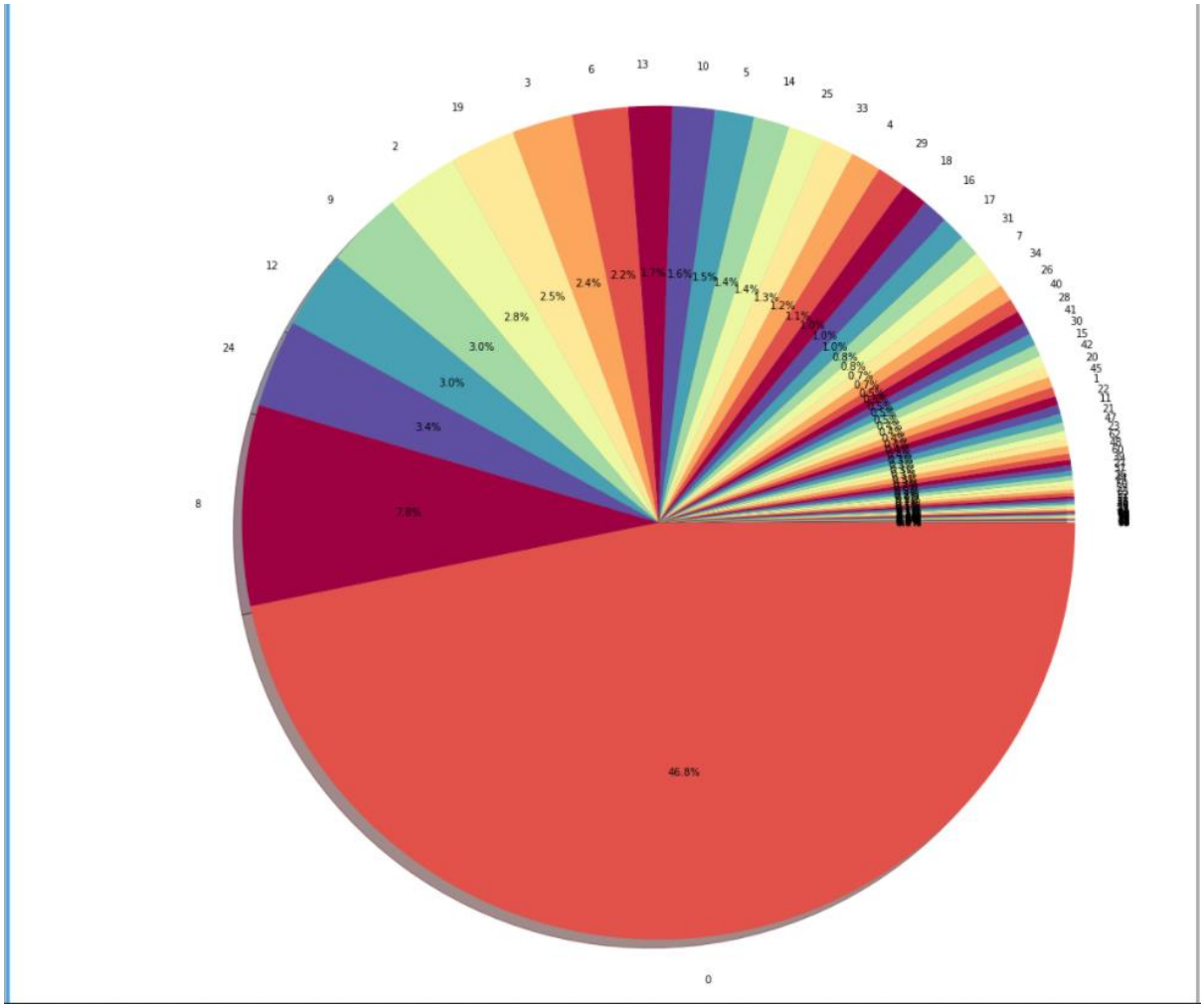
For generating 2-grams we pass the value of n=2 in ngrams function of NLTK.

Trigram or 3 -grams

In case of 3-grams, we pass the value of n=3 in ngrams function of NLTK.

- N-Grams are useful to create features from text corpus for machine learning algorithms like SVM, Naive Bayes, etc.
- N-Grams are useful for creating capabilities like autocorrect, auto completion of sentences, text summarization, speech recognition, etc.

CLASS IMBALANCING - BEFORE UPSAMPLING AND DOWNSAMPLING



- Up sampled minority classes using text augmentation
- Down sampled major class Group 0

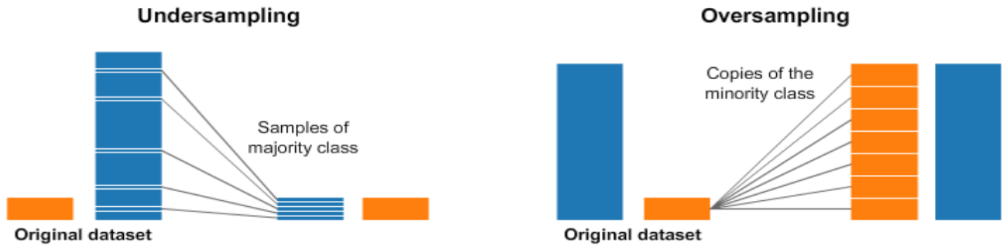
Text Augmentation

Data augmentation techniques are used to generate additional, synthetic data using the data you have.

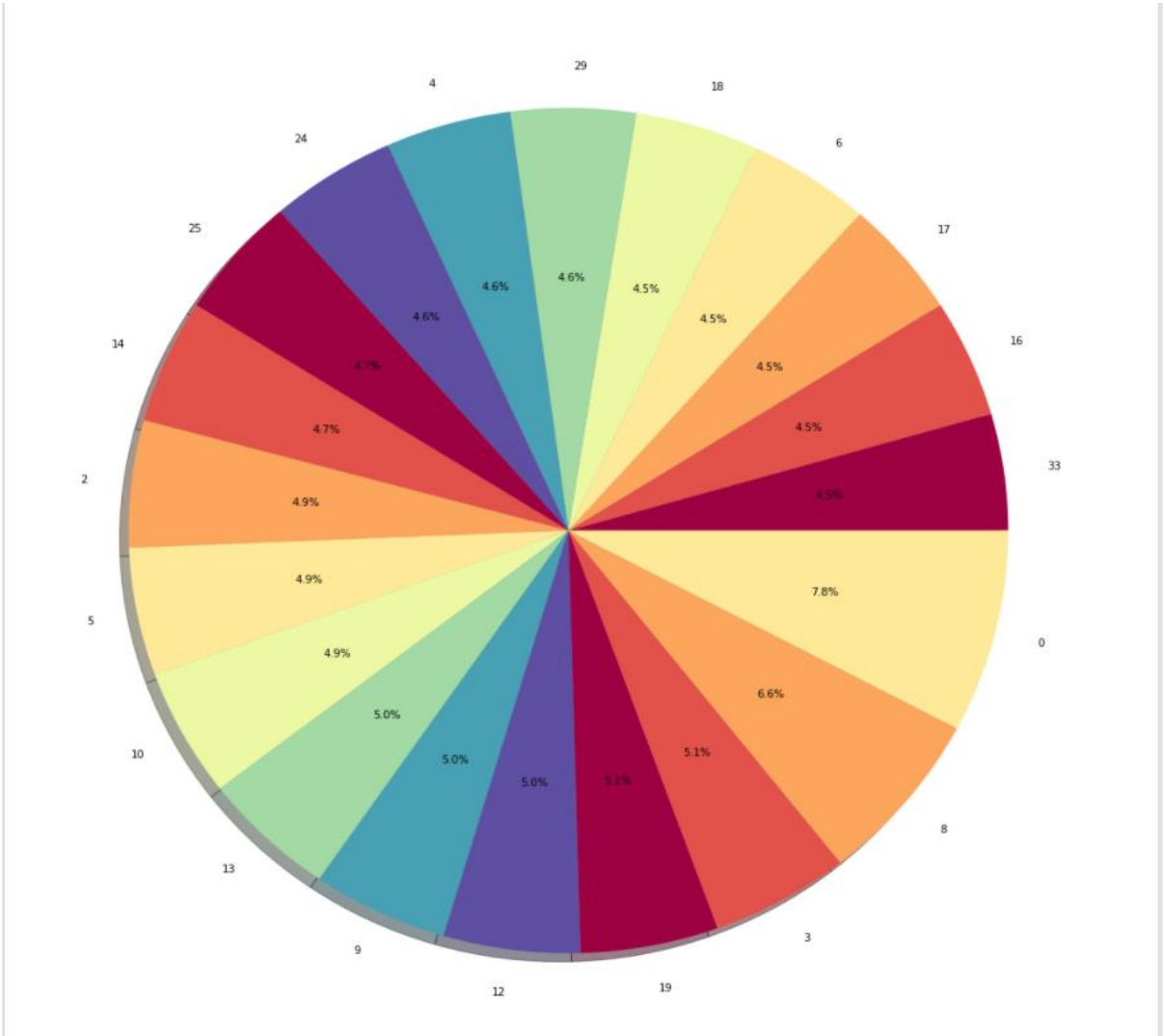
We used Synonym Replacement technique to up sample text

Randomly choose n words from the sentence that are not stop words. Replace each of these words with one of its synonyms chosen at random.

AFTER UPSAMPLING AND DOWNSAMPLING



Under- and Over-Sampling



Count Vectorization

Count Vectorization involves counting the number of occurrences each words appears in a document (i.e distinct text such as an article, book, even a paragraph!). Python's Sci-kit learn library has a tool called CountVectorizer to accomplish this.

Tfidf Vectorization

TfidfVectorizer - Transforms text to feature vectors that can be used as input to estimator.vocabulary_ Is a dictionary that converts each token (word) to feature index in the matrix, each unique token gets a feature index.

Word Embedding

Tokenizing Text -> Representing each word by a number
Mapping of original word to number is preserved in word_index property of tokenizer
Tokenized applies basic processing like changing it to lower case, explicitely setting that as False
Lets keep all news to 300, add padding to news with less than 300 words and truncating long ones

GloVe (Global Vectors) Embedding:

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

ML MODELS

Logistic Regression

classification algorithm to predict multi class labels

- Applied logistic regression after applying count vectorization on data

Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
LR_Count_Vector	0.606164	0.670583	0.604288	0.007527	0.606164	0.648491

- Applied logistic regression after applying TF-IDF Vectorization

	Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	LR_TF-IDF	0.371645	0.402607	0.34853	0.00664	0.371645	0.395404

- Applied logistic regression after applying TF-IDF sequence padding

Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
LR_seq	0.185968	0.366969	0.167797	0.007708	0.185968	0.174948

- Logistic Regression on Original data before Up sampling

	Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	LR_count_vector_org	0.616467	0.881371	0.578948	0.010855	0.616467	0.581318

Naïve Bayes

- Applied Naïve Bayes after applying count vectorization on data

Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
NB_Count_Vectors	0.340926	0.360524	0.319414	0.0092	0.340926	0.377817

- Applied Naïve Bayes after applying TF-IDF Vectorization

	Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	NB_TF-IDF	0.294022	0.316722	0.270852	0.01329	0.294022	0.375193

- Applied Naïve Bayes after applying TF-IDF sequence padding

	Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	NB_seq	0.106019	0.121039	0.079185	0.003835	0.106019	0.195449

- Naïve Bayes on Orginal data before Up sampling

	Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
)	NB_Count_Vectors_org	0.485046	0.588457	0.484405	0.011209	0.485046	0.567751

XGBoost

- Applied Xgboost after applying count vectorization on data

Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
NB_Count_Vectors	0.340926	0.360524	0.319414	0.0092	0.340926	0.377817

- Applied Xgboost after applying TF-IDF Vectorization

	Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	NB_TF-IDF	0.294022	0.316722	0.270852	0.01329	0.294022	0.375193

- Applied Xgboost after applying TF-IDF sequence padding

	Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	NB_seq	0.106019	0.121039	0.079185	0.003835	0.106019	0.195449

DL MODELS

RNN Architecture

RNNs perform the same task for every element of a sequence, with the output being dependent on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far.

```
Model: "sequential_60"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_55 (Embedding)     (None, None, 300)         1740900
_________________________________________________________________
simple_rnn_68 (SimpleRNN)    (None, None, 40)          13640
_________________________________________________________________
simple_rnn_69 (SimpleRNN)    (None, None, 40)          3240
_________________________________________________________________
simple_rnn_70 (SimpleRNN)    (None, None, 40)          3240
_________________________________________________________________
simple_rnn_71 (SimpleRNN)    (None, 40)                3240
_________________________________________________________________
dense_51 (Dense)             (None, 12899)             528859
=================================================================
Total params: 2,293,119
Trainable params: 552,219
Non-trainable params: 1,740,900
_________________________________________________________________
```

RNN Performance

	Model	test_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
RNN_seq	0.082171	0.012479	0.0	0.082171	0.006752

LSTM Architecture

It is special kind of recurrent neural network that is capable of learning long term dependencies in data. This is achieved because the recurring module of the model has a combination of four layers interacting with each other.

```
Model: "sequential_48"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_43 (Embedding)     (None, None, 300)         1740900
_________________________________________________________________
lstm_22 (LSTM)               (None, 32)                42624
_________________________________________________________________
dropout_18 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_39 (Dense)             (None, 12899)             425667
=================================================================
Total params: 2,209,191
Trainable params: 468,291
Non-trainable params: 1,740,900
_________________________________________________________________
```

LSTM Performance

	Model	test_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	LSTM_c	0.082171	0.012479	0.0	0.082171	0.006752

GRU Architecture

To solve the vanishing gradient problem of a standard RNN, GRU uses, so-called, **update gate and reset gate**. Basically, these are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.

```
Model: "sequential_59"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_54 (Embedding)     (None, None, 100)         580300
_________________________________________________________________
gru_9 (GRU)                  (None, 32)                12864
_________________________________________________________________
dropout_29 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_50 (Dense)             (None, 12899)             425667
=================================================================
Total params: 1,018,831
Trainable params: 1,018,831
Non-trainable params: 0
_________________________________________________________________
```

GRU Performance

	Model	test_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	GRU_WE	0.082558	0.013249	0.00077	0.082558	0.028075

-

ML Model performance

	Model	test_accuracy_mean	train_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	LR_Count_Vector	0.606164	0.670583	0.604288	0.007527	0.606164	0.648491
0	LR_count_vector_org	0.616467	0.881371	0.578948	0.010855	0.616467	0.581318
0	NB_Count_Vectors_org	0.485046	0.588457	0.484405	0.011209	0.485046	0.567751
0	LR_TF-IDF	0.371645	0.402607	0.34853	0.00664	0.371645	0.395404
0	NB_Count_Vectors	0.340926	0.360524	0.319414	0.0092	0.340926	0.377817
0	NB_TF-IDF	0.294022	0.316722	0.270852	0.01329	0.294022	0.375193
0	LR_seq	0.185968	0.366969	0.167797	0.007708	0.185968	0.174948
0	NB_seq	0.106019	0.121039	0.079185	0.003835	0.106019	0.195449

DL Model performance

	Model	test_accuracy_mean	test_f1-score_mean	test_f1-score_std	test_recall_score_mean	test_precision_score_mean
0	GRU_WE	0.082558	0.013249	0.00077	0.082558	0.028075
0	RNN_WE	0.082171	0.012479	0.0	0.082171	0.006752
0	RNN_seq	0.082171	0.012479	0.0	0.082171	0.006752
0	LSTM_c	0.082171	0.012479	0.0	0.082171	0.006752
0	CNN_GRU_WE	0.082171	0.012479	0.0	0.082171	0.006752

Approaches to improve model

GRU, LSTM and RNN returns very less f1-score and less accuracy

Other models like BERT will be tried out to improve performance

Hyper tuning in Deep learning models will be focused in next iteration

Grid search will be used in Hyper tuning to increase score metrics