INTERIM REPORT – TEMPLATE

Your interim report is also the milestone-1. It should Summarize the 1st 3 deliverables mentioned in the project document. The report structure and brief requirements of each section is listed below.

1. Summary of problem statement, data and findings Every good abstract describes briefly what was intended at the outset, and summarizes findings and implications.

A ticketing system is a customer service tool that helps companies manage their service and support cases. The system or app creates a "ticket" which documents customer requests and interactions over time, making it easier for customer service reps to resolve complicated issues. It is a tool that allows IT support to be organized, focused, efficient. This directly impacts costs and revenues, customer retention, and public brand image.

An incident is something that is unplanned interruption to an IT service or reduction in the quality of an IT service that affects the Users and the Business. The main goal of Incident Management process is to provide a quick fix / workarounds or solutions that resolves the interruption and restores the service to its full capacity to ensure no business impact. It goes through a lifecycle of Identifying and recording the issue to management and resolution of the same. Assigning the incidents to the appropriate person or unit in the support team has critical importance to provide improved user satisfaction while ensuring better allocation of support resources.

An incident management system guided by powerful AI techniques that can classify incidents to right functional groups can help organizations to reduce the resolving time of the issue and yield more productive tasks. In this capstone project, the goal is to build a classifier that can classify the tickets by analyzing text.

Understanding and visualization of data

Data Cleaning and feature extraction

Model fitting and hyperparameter tuning

Effect of word embeddings on performance of model

Our approach consisted of 4 broad steps:

1. Exploratory data analysis: Data was explored and various visualizations were created using word cloud to get a sense of terminology and to identify stopwords and special characters that might need cleaning
2. Data Cleaning and feature vector generation: Upon identification of stopwords, punctuations, special characters and missing fields, data was appropriately cleaned and word frequency and TF-IDF was determined to produce feature vector for model fitting.
3. Model fitting and hyperparameter tuning: input vector was fit into traditional models like Naive Bayes, logistic Regression, SVM, Random Forest, XG boost, KNN and simple Neural Network. Random forest and Neural Network gave better accuracy than others.Complex network consisting of Dense, Batch Normalization and Dropout layer was constructed for modelfitting. The ones that performed better than others were further tuned through their hyperparameters.
4. Word embeddings boost to model performance: Different Embeddings namely ELMO, BERT, FastText, Word2Vec and Glove were explored to further enhance the model performance.

A ticket after being raised will go through the flow as shown in flowchart below:

A ticket is raised. Data will consist of Short Description, Description and Caller details

Description is converted to string and is taken up for further processing

If Size of Description is greater than that of Short description?

Description is replaced by Short Description

Description is converted to lower case

Following is removed form Description:

Line breaks, ‘\n’,Emails, Underscore, Punctuation, Additional Whitespaces,

Digits other than with format Job\_<jobnumber>,

Stopwords, Caller names

Extract bigrams and trigrams

Determine TF-IDF for unigrams, bigrams, trigrams

Based on TF-IDF values, transform unigrams, bigrams and trigrams to feature vectors

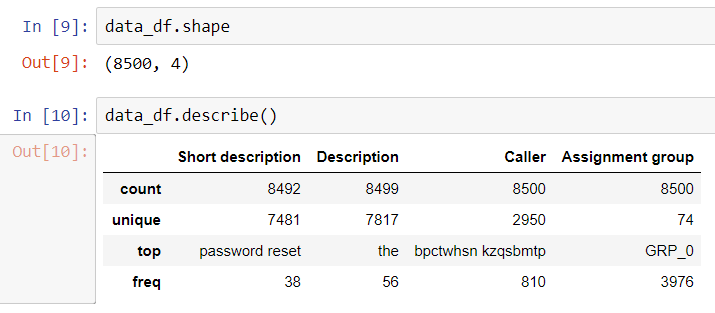
Pass generated vector through pre-trained model

Load word embeddings

Create embedding matrix

Output

Following is an image summarizing data provided to us for building classifier:

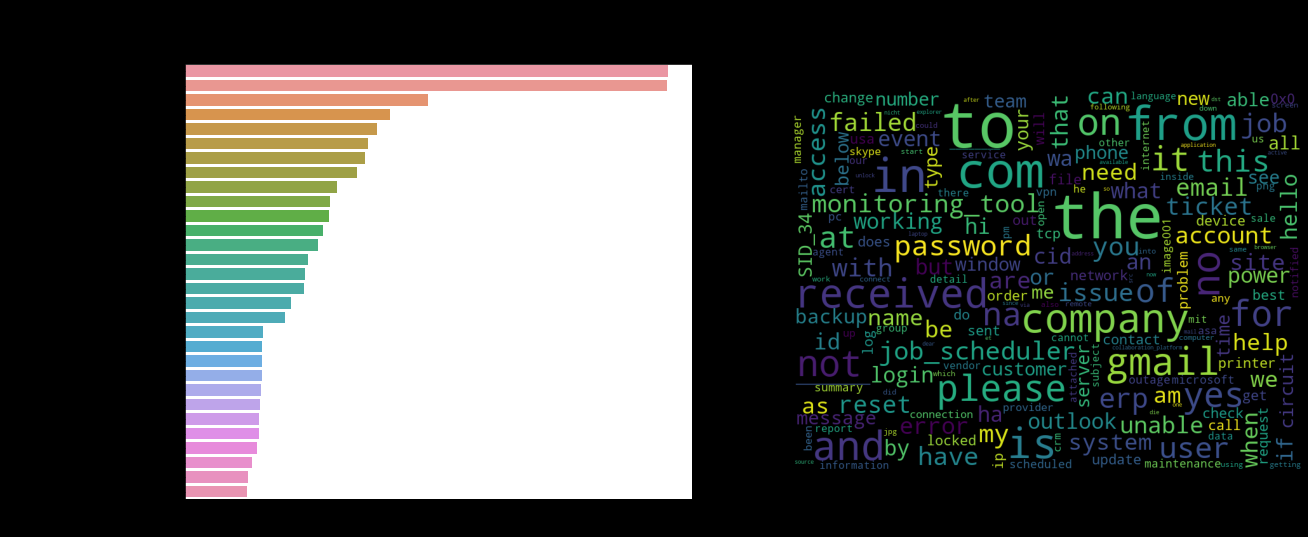


Data is of shape 8500 \* 4.

8500 entries. With 4 columns namely

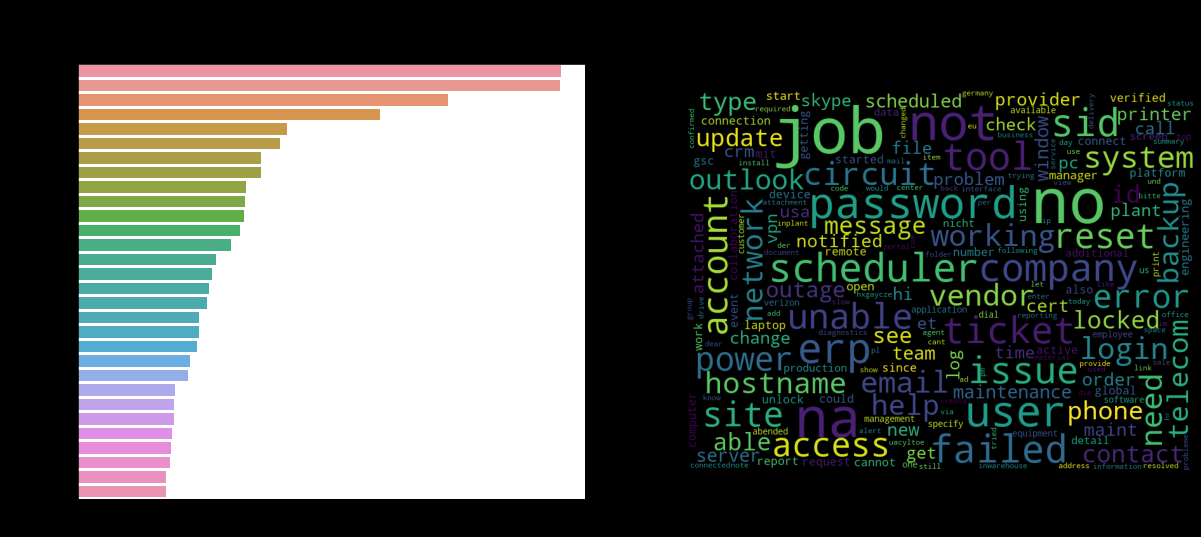
1. ‘Short description’: Short summary of ticket
2. ‘Description’: Description of the issue faced for which the ticket is raised. Since, this column has more information that ‘Short description’column, this will be our input variable. There are 7817 unique values on the description. This signifies there might be some duplicates in the column. There are cases where length of Description is smaller than that of ‘Short Description’column. In such cases, Short description will be considered instead of Description.
3. ‘Caller’: Name of the person who raised the ticket. There are 2950 unique callers. This column does not provide any special information in differentiating the Assignment group allotment. Hence, this column can be removed for dataset.
4. ‘Assignment\_group’: Group name to which the ticket is assigned to.There are total of 74 groups out of which maximum entries belong to GRP\_0. out of 8500, 3976 entries belong to GRP\_0. This column will be our target variable.

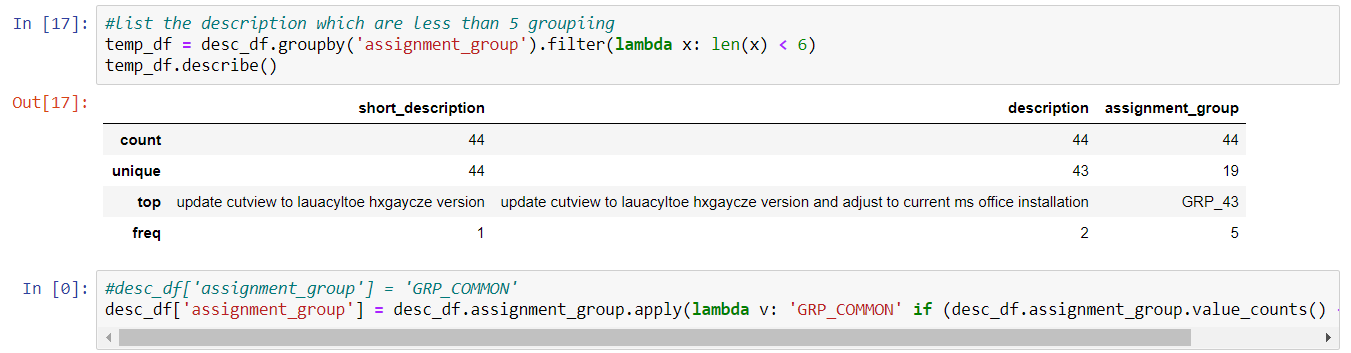
First look of Description column:



There are os related new line and line termination tags. Few description have header - received from : - which doesn't provide much information on classification. Few description also have footer note - Thanks/regards followed by name - which doesn't provide much information on classification. There are few encoded words in description which could be name of the persons that are encrypted owing to PII governance which needs to be handled in pre-processing. There are also few system drive path, software versioning number and ip addresses. Some description might have attached with evidence photos which resulted in cid: tags in the description footer.

Word cloud for Description column upon cleaning:





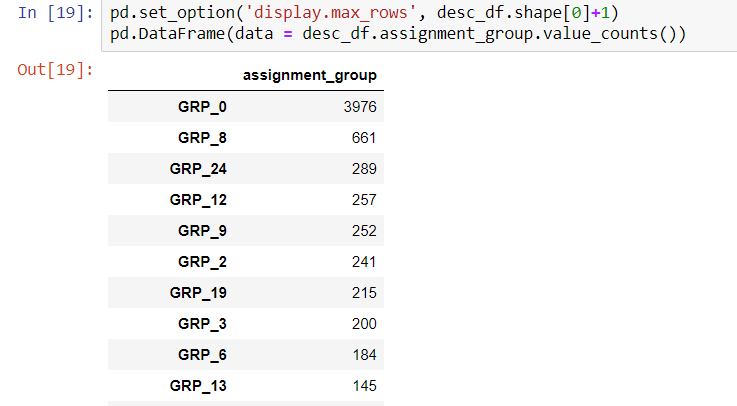
There are few groups that have just one entry. It will be difficult to train a model in presence of such low entries. Hence, groups less than 6 entries are clubbed together to form just one group. There were 44 such entries. And all these are clubbed together under one group GRP\_COMMON.

#### ****Data Cleaning Analysis****

#### Following are the preprocessing steps taken on data:

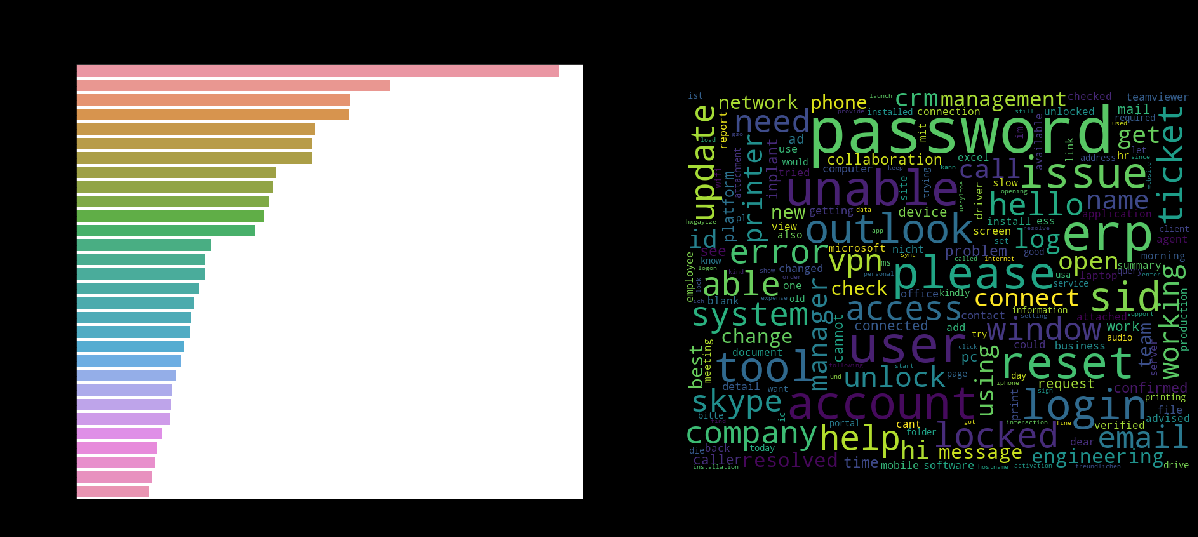
1. Escape characters like new line are removed
2. Stopwords are updated to include received from header & footer tags and image attachment reference in footer
3. email id references are removed
4. Punctuation characters are removed
5. Stop words are removed
6. Check frequent words with respect to problem context
7. Digits other than Job\_<job number> are removed
8. Caller names are removed

Following is the list of top 10 Assignment groups:

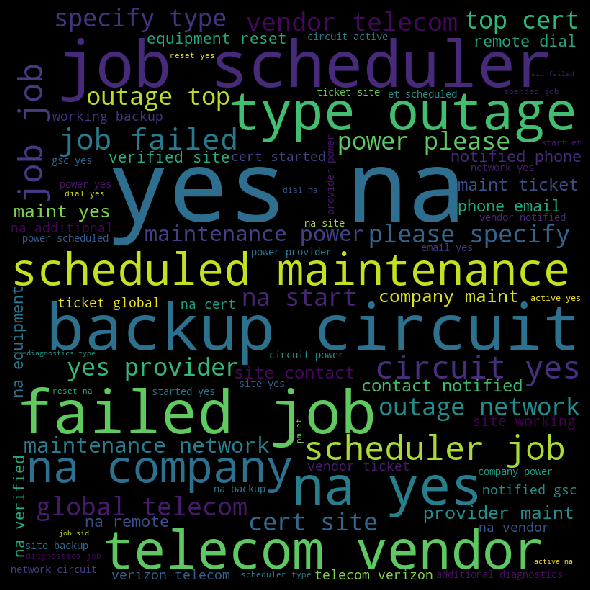


Wordclouds for top 10 groups:

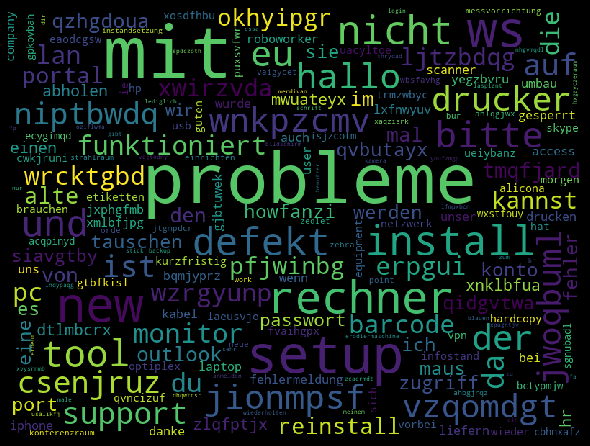
GRP\_0



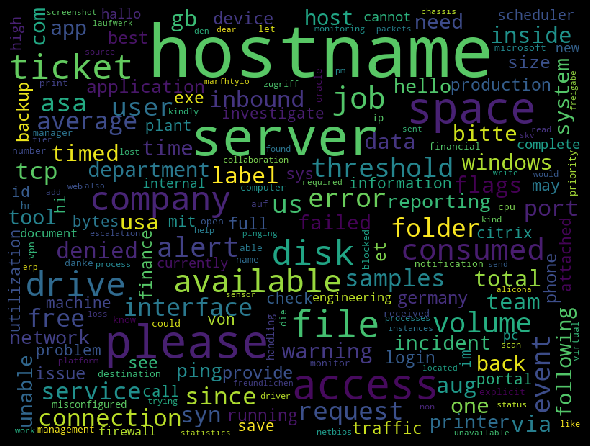
GRP\_8



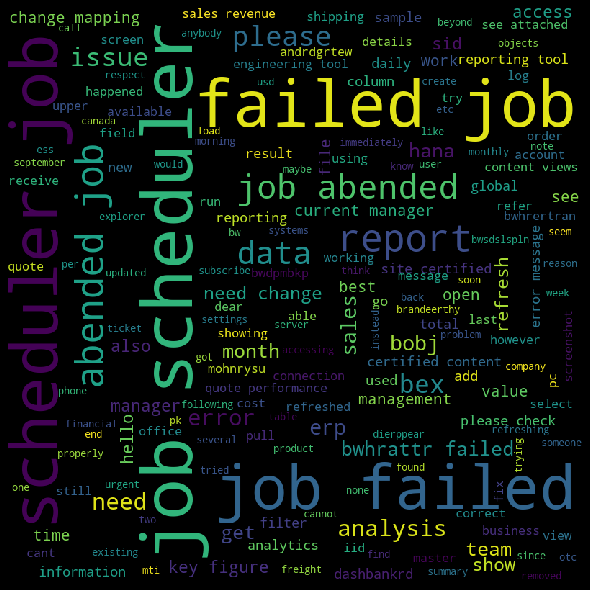
GRP\_24



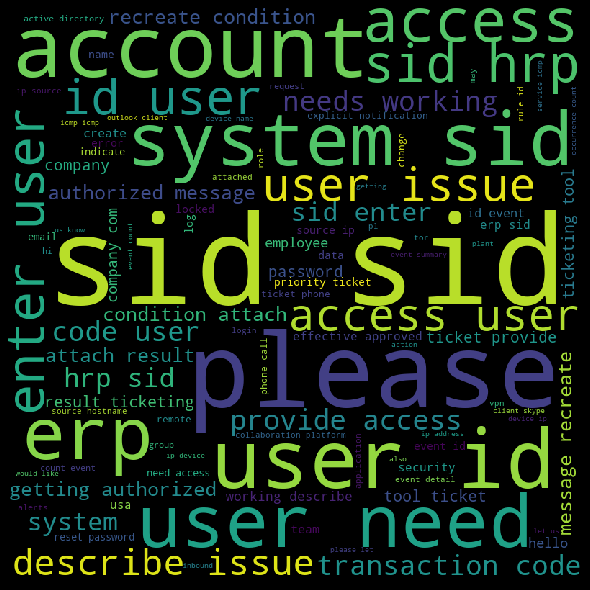
GRP\_12



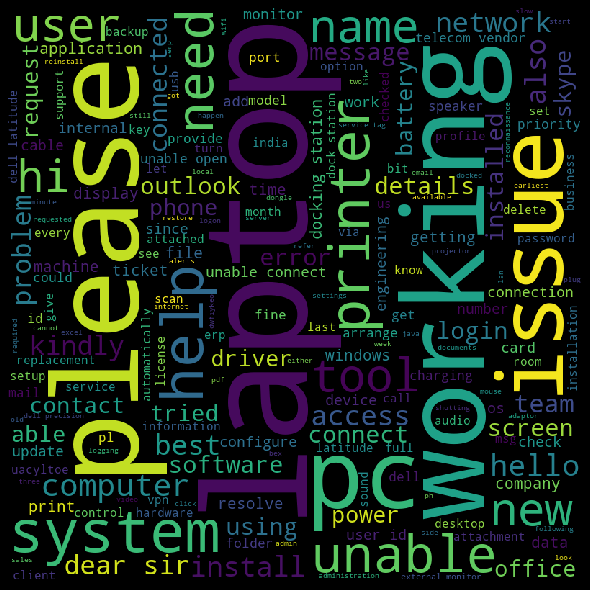
GRP\_9



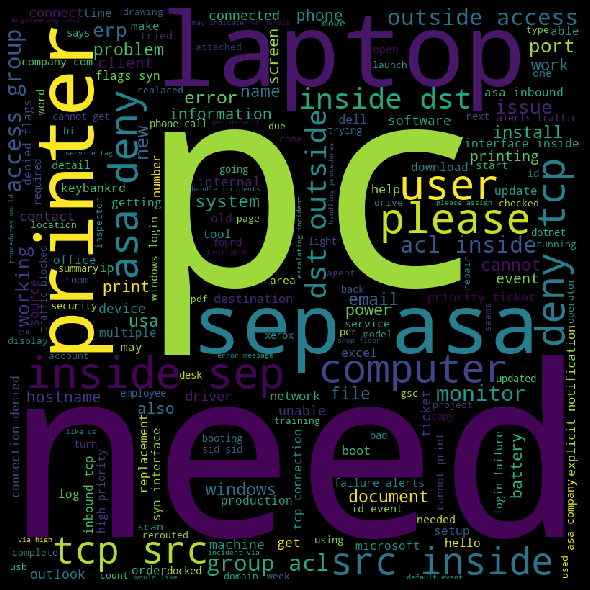
GRP\_2



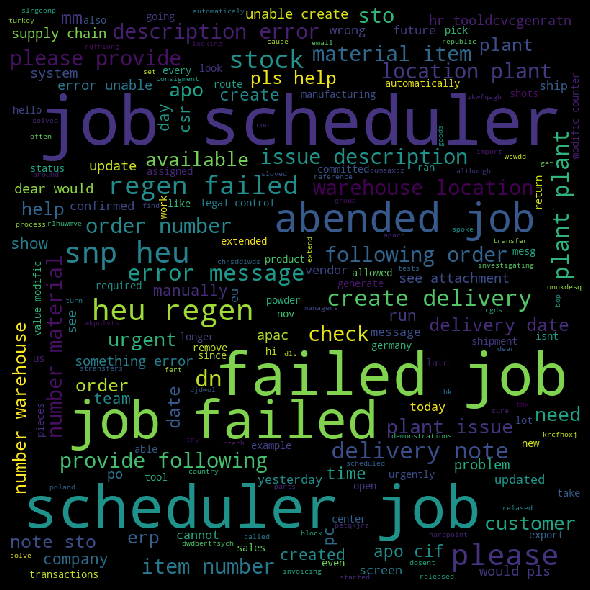
GRP\_19



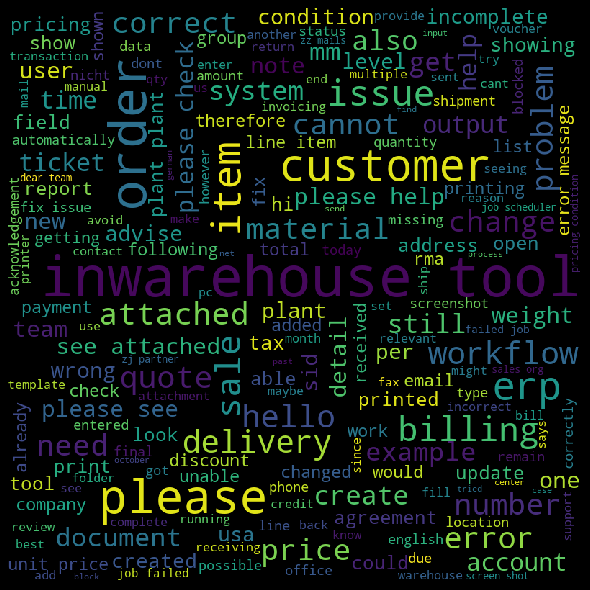
GRP\_3



GRP\_6



GRP\_13



**N gram analysis**

**Word cloud for unigram:**

Total no of uni-grams from corpus without stopwords : 14520

Size of uni-gram Vocabulary upon cleaning : 3049

Size of updated uni-gram Tokens based on frequency : 2659



Word Cloud for bigrams:

Total no of bi-grams from corpus without stopwords : 61580

Size of bi-gram Vocabulary upon cleaning : 30368

Size of updated bi-gram Tokens based on frequency: 26692



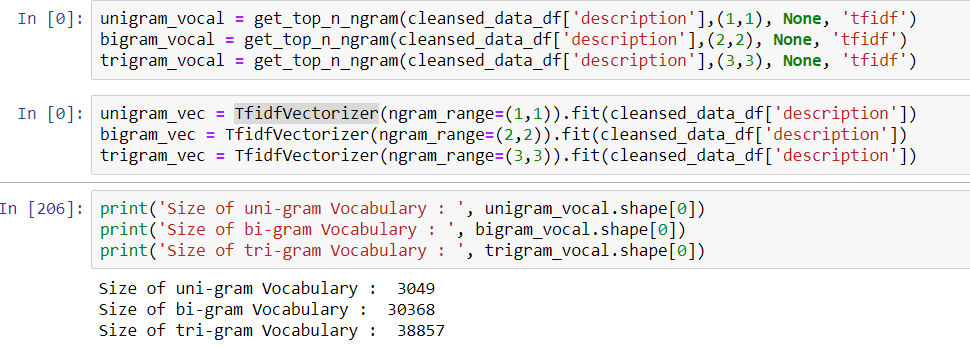
Word Cloud for trigrams:

Total no of tri-grams from corpus without stopwords: 75257

Size of tri-gram Vocabulary upon cleaning : 38857

Size of updated tri-gram Tokens based on frequency : 36155

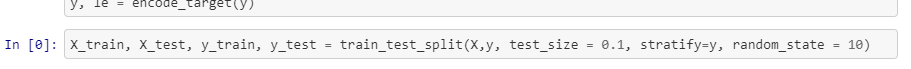


After Generation of Ngrams, TF-IDF was calculated and on its basis, data was further vectorized and input matrix consisting of unigrams, bigrams and trigrams was constructed. <removal of high and low frequency words> 

3. Deciding Models and Model Building Based on the nature of the problem, decide what algorithms will be suitable and why? Experiment with different algorithms and get the performance of each algorithm.

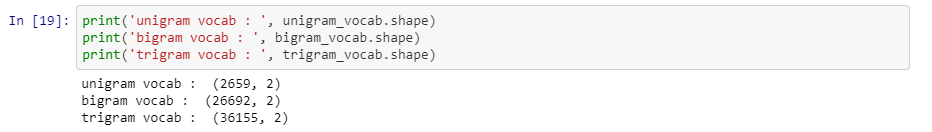
After cleansing the data, we tried different machine learning models namely, Naïve Bayes, Logistic Regression, SVM Classifier, Random Forest, XGBoost and KNN.

For modeling we have taken 90% train and 10% test data for modeling.

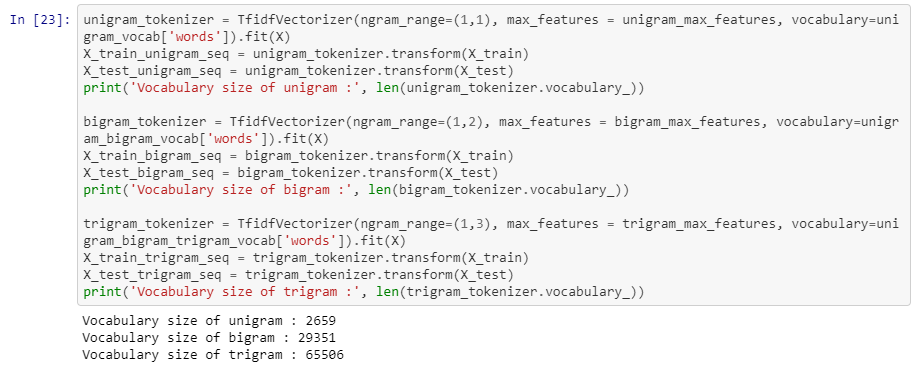


Since it is a classification problem we have used the aforementioned algorithms.

Modeling was done on uni-gram, bi-gram and tri-grams. Below is the vocab size for all the three.



Then we calculated the Tf-Idf Vectorizer for Uni-gram, bi-gram and tri-gram.



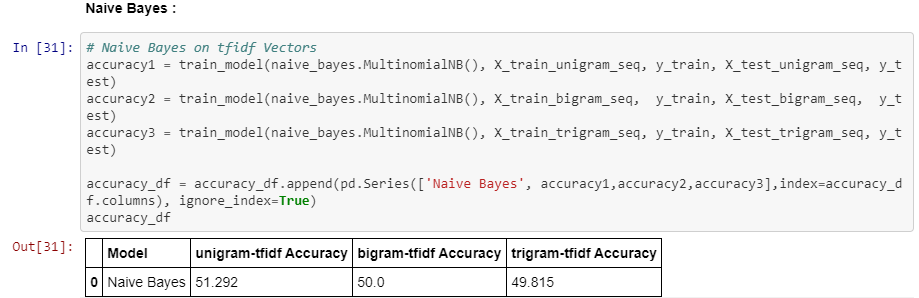
We then modeled the uni-gram, bi-grams and tri-grams vectors.

Below are the models used and their corresponding accuracy scores.

**Naïve Bayes**

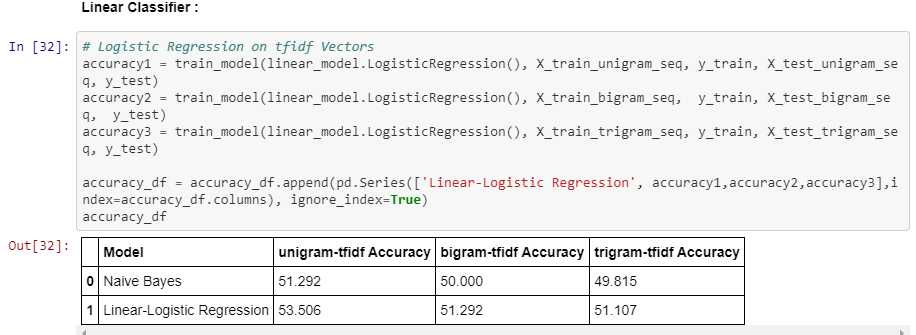
Naïve Bayes is a fast algorithm for classification problem. This algorithm is a good fit for real-time prediction, multi-class prediction, recommendation system, text classification, and sentiment analysis use cases. Naive Bayes Algorithm can be built using Gaussian, Multinomial and Bernoulli distribution. This algorithm is scalable and easy to implement for the large data set.

Due to its better performance with multi-class problems and its independence rule, Naive Bayes algorithm perform better or have a higher success rate in text classification, Therefore, it is used in [Sentiment Analysis](https://www.educba.com/sentiment-analysis-social-media/) and Spam filtering.



**Logistic Regression**

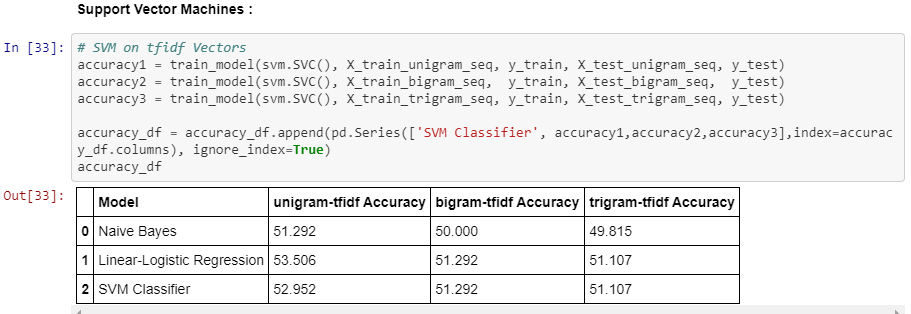
Logistic Regression is widely used technique because it is very efficient, does not require too many computational resources, it’s highly interpretable, it doesn’t require input features to be scaled, it doesn’t require any tuning, it’s easy to regularize, and it outputs well-calibrated predicted probabilities. Another advantage of Logistic Regression is that it is incredibly easy to implement and very efficient to train.



**SVM Classifier**

SVM separates data points using a hyperplane with the largest amount of margin. That's why an SVM classifier is also known as a discriminative classifier. SVM finds an optimal hyperplane which helps in classifying new data points. The benefit is that it can capture much more complex relationships between data points without having to perform difficult transformations on your own.

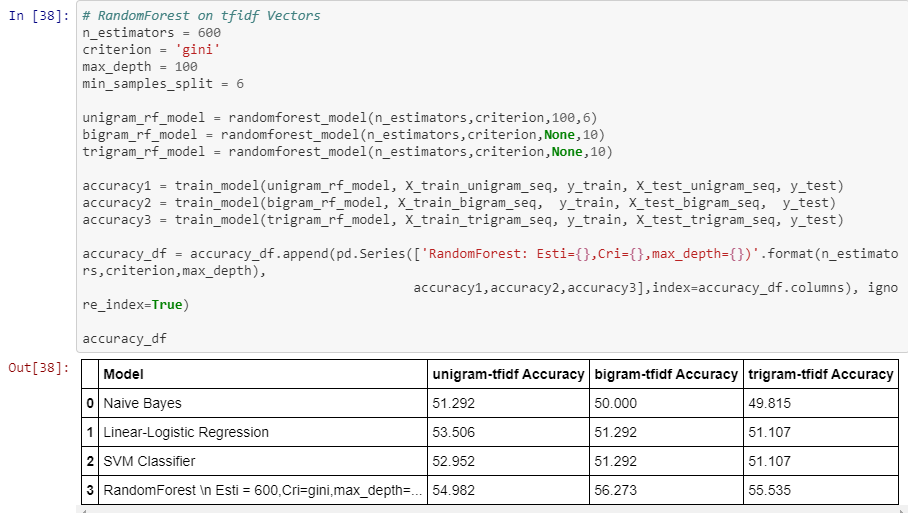
SVM's can model non-linear decision boundaries, and there are many kernels to choose from. They are also fairly robust against overfitting, especially in high-dimensional space.



**Random Forest**

Random forest is considered as a highly accurate and robust method because of the number of decision trees participating in the process. It does not suffer from the overfitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.

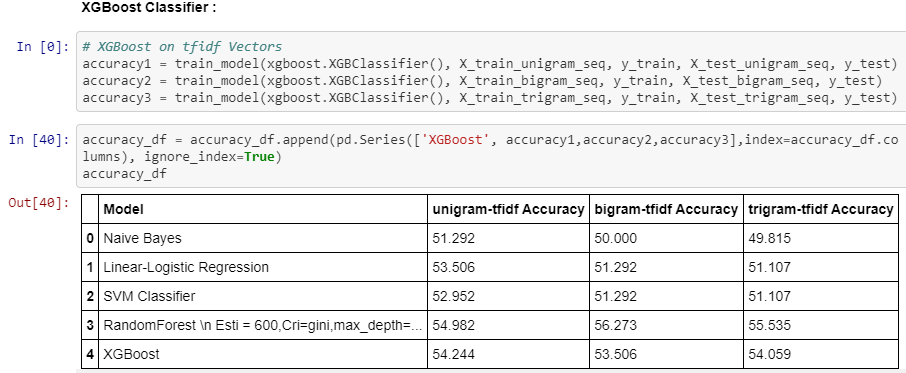
Random forests can also handle missing values. There are two ways to handle these: using median values to replace continuous variables, and computing the proximity-weighted average of missing values.



**XGBoost**

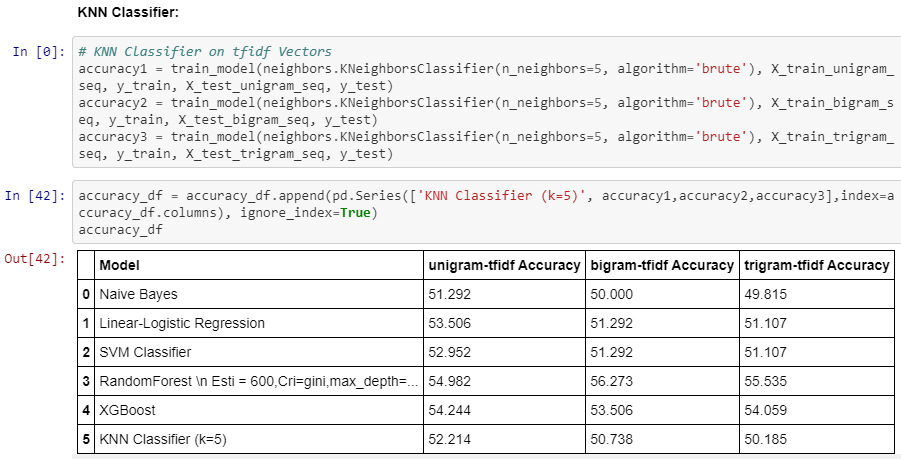
The XGBoost library implements the [gradient boosting decision tree algorithm](https://en.wikipedia.org/wiki/Gradient_boosting). XGBoost goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines.

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. Generally, XGBoost is fast when compared to other implementations of gradient boosting. XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems.



**KNN**

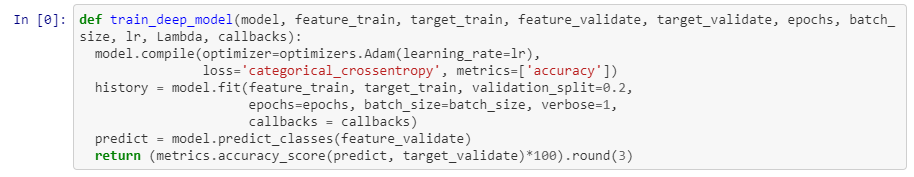
KNN uses lazy training which means all computation is deferred till prediction. This works very well if we have good training data. Naïve Bayes is a quick classifier and K-NN should be preferred when the data-set is relatively small. For this particular data set the KNN was not performing well.

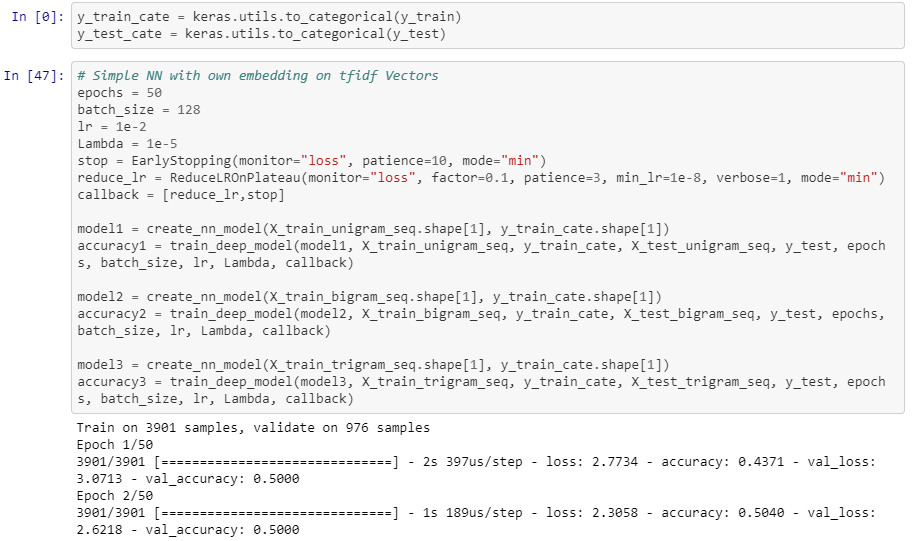


Naïve Bayes performed the worst Random Forest performed the best amongst all classifier.

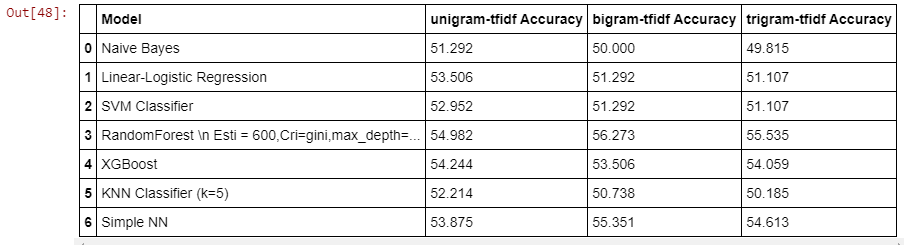
After Machine learning models, we tried deep learning models. Initially we tried simple neural network.







The final accuracy score of the models are below:

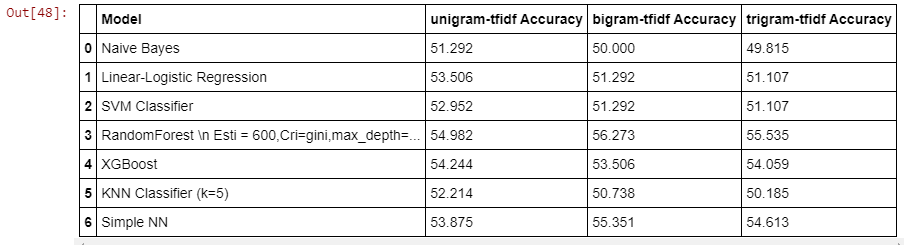


Random Forest performed the best amongst the basic Deep Learning and Machine Learning models.

4. How to improve your model performance? What are the approaches you can take to improve your model? Can you do some feature selection, data manipulation and model improvements? Provide your code and as much as visualizations you can share to describe what you have done so far.

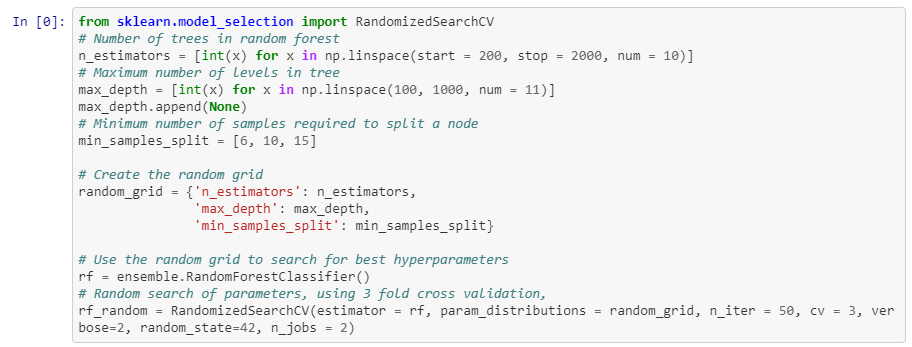
We dropped columns “caller” as it was not helpful for modeling. As mentioned in above section, we removed all the stop words and cleansed the data and combined the rows where there were few data sets and then did modeling on that. However, the results were not great. The accuracy score was ~54%.

For better performance of the model we tested for uni-gram, bi-gram and tri-gram vectors.



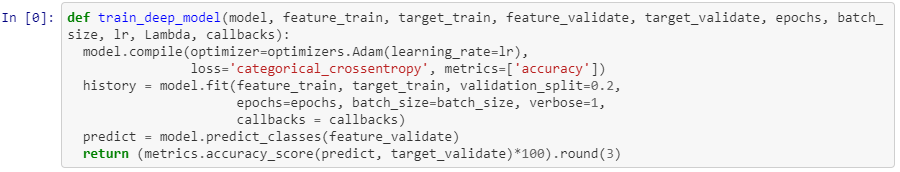
Random Forest and Simple Neural Network has better accuracy score in bi-gram and tri-grams. However, the others performed well in uni-grams.

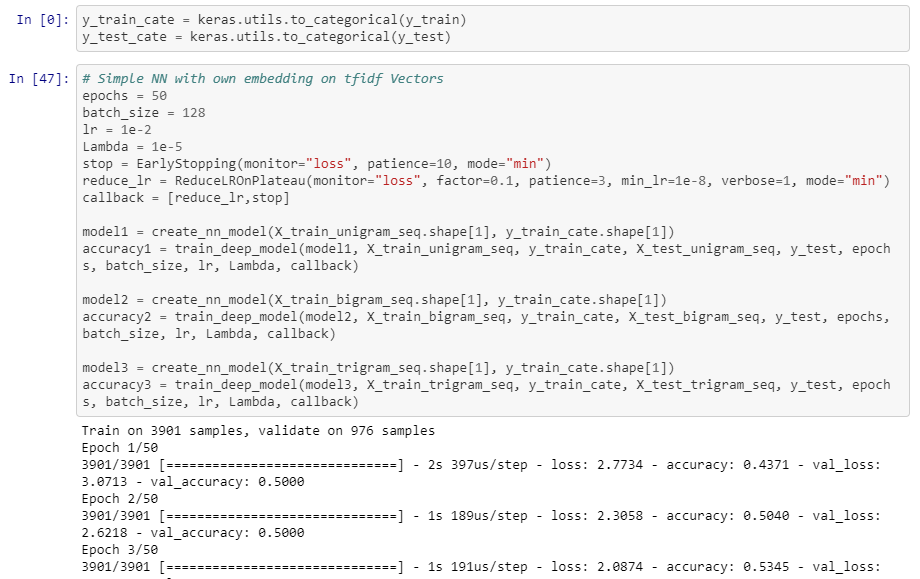
We performed hyper-parameter tuning for Random Forest to find best parameters.



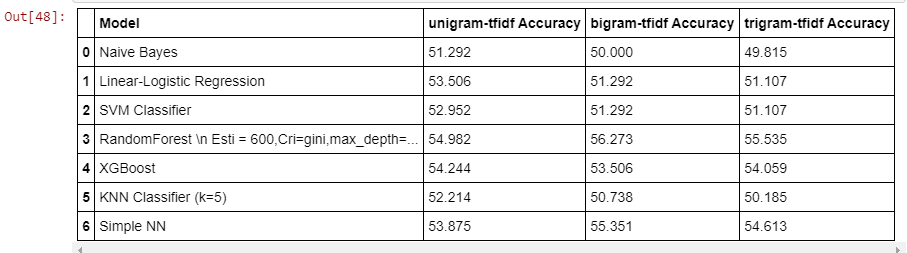
Initially the neural network score was 50.526, 52.105 and 53.333 for uni-gram, bi-gram and tri-grams. To improve the model score we added more Dense Layer, performed batch normalization, added Dropout Layer, used callback and optimizer regularization loss.







We ran the model for 50 epochs. Below are the accuracy score after performing the above mentioned steps.

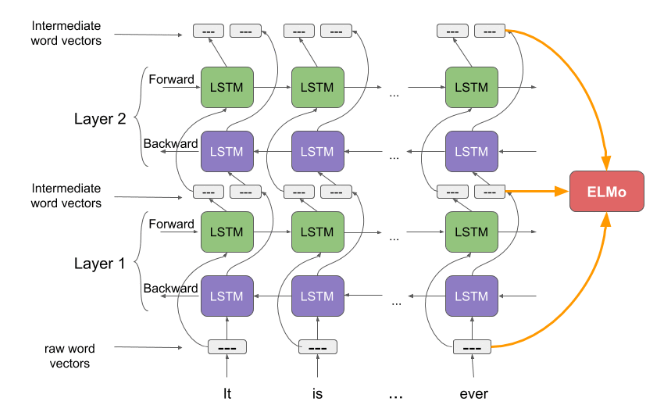


**Word Embeddings:**

**ELMO:**

ELMO is a novel way to represent words in vectors or embeddings. These word embeddings are helpful in achieving state-of-the-art (SOTA) results in several NLP tasks:

ELMo word vectors are computed on top of a two-layer bidirectional language model (biLM). This biLM model has two layers stacked together. Each layer has 2 passes — forward pass and backward pass:



The architecture above uses a character-level convolutional neural network (CNN) to represent words of a text string into raw word vectors. These raw word vectors act as inputs to the first layer of biLM. The forward pass contains information about a certain word and the context (other words) before that word. The backward pass contains information about the word and the context after it. This pair of information, from the forward and backward pass, forms the intermediate word vectors. These intermediate word vectors are fed into the next layer of biLM. The final representation (ELMo) is the weighted sum of the raw word vectors and the 2 intermediate word vectors.

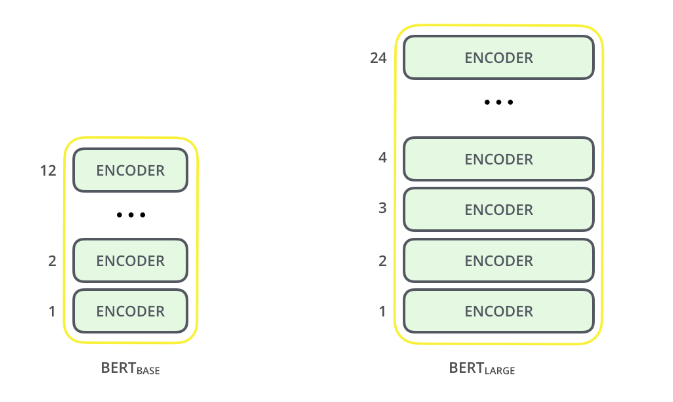
**BERT:**

First, it’s easy to get that BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Each word here has a meaning to it and we will encounter that one by one in this article. For now, the key take away from this line is – **BERT is based on the Transformer architecture.**

Second, BERT is pre-trained on a large corpus of unlabelled text including the entire Wikipedia(that’s 2,500 million words!) and Book Corpus (800 million words). Third, BERT is a **“deeply bidirectional”** model. Bidirectional means that BERT learns information from both the left and the right side of a token’s context during the training phase.

The bidirectionality of a model is important for truly understanding the meaning of a language. We can fine-tune it by adding just a couple of additional output layers to create state-of-the-art models for a variety of NLP tasks. The BERT architecture builds on top of Transformer. We currently have two variants available:

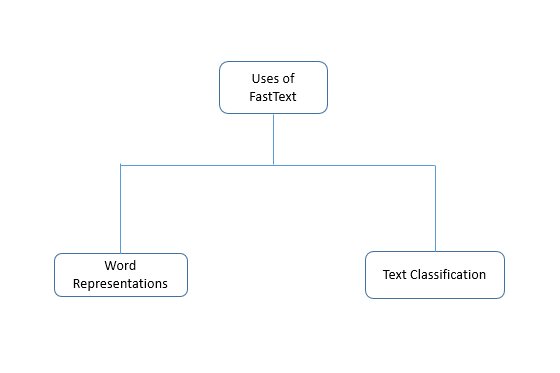
* BERT Base: 12 layers (transformer blocks), 12 attention heads, and 110 million parameters
* BERT Large: 24 layers (transformer blocks), 16 attention heads and, 340 million parameters

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/bert_encoder.png)

**Fast Text:**

What is FastText?

FastText is a library created by the Facebook Research Team(FLAIR) for efficient learning of word representations and sentence classification.



**Word2Vec:**

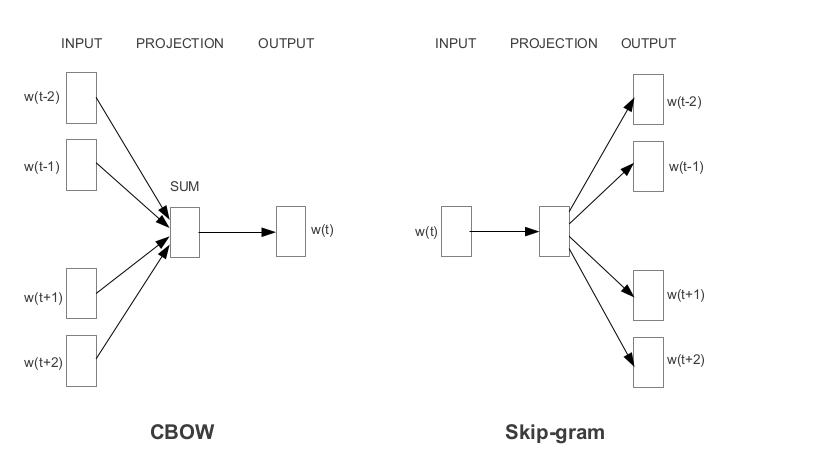
Word2Vec is one of the most popular pretrained word embeddings developed by Google. Word2Vec is trained on the Google News dataset (about 100 billion words).

The architecture of Word2Vec is really simple. It’s a feed-forward neural network with just one hidden layer. Hence, it is sometimes referred to as a **Shallow Neural Network architecture**.

Word2Vec is classified into two approaches:

* Continuous Bag-of-Words (CBOW)
* Skip-gram model

Continuous Bag-of-Words (CBOW) model learns the focus word given the neighbouring words whereas the Skip-gram model learns the neighbouring words given the focus word.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/Screenshot-from-2020-03-12-13-05-42.png)

**GloVe:**

The basic idea behind the GloVe word embedding is to derive the relationship between the words from Global Statistics.

One of the simplest ways is to look at the co-occurrence matrix. A co-occurrence matrix tells us how often a particular pair of words occur together. Each value in a co-occurrence matrix is a count of a pair of words occurring together.

One of the simplest ways is to look at the co-occurrence matrix. A co-occurrence matrix tells us how often a particular pair of words occur together. Each value in a co-occurrence matrix is a count of a pair of words occurring together.