**Chapter 1. Use Case Demo Model**

A screenshot of a cell phone

Description automatically generated

**Illustration 1 showing use-case demo architecture**

The Illustration 1 shows the Use-Case Demo Architecture representing a workflow using real time social media data (Twitter) to predict depression in users as well as understanding the sentiment analysis of the real time data extracted from social media platform (Twitter). The steps involved at each stage will be discussed further in this section. In addition to it, the steps involved in transformation of the final matrix are represented in detail under Illustration 2.

**1.1 Importing Libraries**

To begin with, the first stage involves importing some essential libraries and functions as shown

in Table 1(a) and 1(b). These libraries and functions will help to perform various statistical and

visual operations on the dataset required for pre-processing and classification of desired results.

|  |
| --- |
| 1. import pandas as pd 2. import nltk 3. from nltk.corpus import stopwords #importing stopwords from nltk 4. stop\_words = stopwords.words('english') #loading stop words 5. import numpy as np #numpy for matrix manipulation 6. from sklearn.feature\_extraction.text import TfidfVectorizer #load tfidf vector 7. import re #import regular expression for text manipulation 8. from sklearn.metrics import classification\_report, accuracy\_score 9. from sklearn import metrics, svm 10. from nltk.stem.snowball import SnowballStemmer 11. from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer 12. import nltk 13. import csv 14. import os 15. import conda 16. from nltk.stem import PorterStemmer 17. conda\_file\_directories = conda.\_\_file\_\_     1. conda\_dirs\_ = conda\_file\_directories.split('lib')[0]     2. proj\_lib\_conda = os.path.join(os.path.join(conda\_dirs\_, 'share'), 'proj')     3. os.environ["PROJ\_LIB"] = proj\_lib\_conda 18. import folium 19. from pprint import pprint 20. import gensim 21. import seaborn as sns 22. from sklearn.model\_selection import train\_test\_split 23. from sklearn.metrics import plot\_confusion\_matrix 24. from sklearn.ensemble import GradientBoostingClassifier 25. import gensim.corpora as corpora 26. from gensim.utils import simple\_preprocess 27. from gensim.models import CoherenceModel 28. import math 29. from keras.models import Sequential 30. from keras.layers import Dense 31. from keras.layers import LSTM 32. from sklearn.preprocessing import MinMaxScaler 33. from sklearn.metrics import mean\_squared\_error 34. from sklearn.metrics import mean\_absolute\_error 35. from keras.preprocessing.sequence import pad\_sequences 36. import itertools 37. from keras.preprocessing.text import Tokenizer 38. from keras.utils.vis\_utils import model\_to\_dot 39. from sklearn.model\_selection import cross\_val\_score |

**Table 1(a) Showing pattern of important libraries imported**

|  |
| --- |
| 1. from sklearn.metrics import precision\_score 2. stemmer = SnowballStemmer("english") 3. from bs4 import BeautifulSoup 4. import string 5. from collections import Counter 6. from scipy import sparse 7. import sklearn 8. import descartes 9. import pyLDAvis.gensim 10. from scipy.sparse import csr\_matrix 11. from IPython.display import SVG 12. from keras.models import Model, Sequential 13. from keras.callbacks import EarlyStopping, ModelCheckpoint 14. from keras.layers import Conv1D, Dense, Input, LSTM, Embedding, Dropout, Activation,GlobalMaxPool1D, Bidirectional, MaxPooling1D #For Neural Networks 15. import spacy 16. import pyLDAvis 17. from sklearn.ensemble import AdaBoostClassifier 18. from sklearn.ensemble import RandomForestClassifier 19. from sklearn.metrics import accuracy\_score 20. import itertools 21. from sklearn.metrics import confusion\_matrix,recall\_score 22. import xgboost as xgb 23. import time 24. from sklearn.linear\_model import LogisticRegression 25. import matplotlib.pyplot as plt |

**Table 1(b) Showing pattern of important libraries imported**

Table 1(a) and 1(b) shows various libraries and functions required for use-case architecture

(discussed in chapter 5)

**1.2 Loading Dataset and Data Cleaning**

The steps involved in this section are discussed in detail under section 5.3.2 and 5.3.3 (chapter 5). However, after loading the dataset, the target variable is defined as ‘0’ for Non-Depressed Tweet and ‘1’ for Depressed Tweet. Further, the data cleaning is performed on the dataset using following operations.

(i) Removal of Punctuation Marks

(ii) Removal of HTML Tags

(iii) Removal of Stop Words

(iv) Removal of URL

(v) Removal of Emojis and extra words

**1.3 Steps involved in the workflow after data cleaning**

# (Step 1. A) TF-IDF (Finding Wordcounts and Frequencies): This step calculates the frequency of each word, as well as the length of total and the unique words in a dataset. The TF-IDF is calculated by TF \* IDF, where

# Term frequency of a word = frequency of word in the document. [62][69]

# Inverse document frequency of a word = log (number of documents )/(number of times word appears in all documents)

# (Step 1. B) LDA (Topic Modeling) and Dominant Topics in Tweets: Latent Dirichlet Allocation (LDA) works on the assumption that the topics present within the corpus are generated from a mixture of topics. These topics further produce words depending on their probability distribution within the corpus. It is basically a technique incorporating the concept of matrix factorization. This step will output distribution of most relevant terms according to percentage of tokens passed for each topic. After the topic distribution, the outcomes for the topics with higher percentage of contribution are calculated using Dominant Topics in Tweets.

# (Step 1. C) NER (Named Entity Recognition) and POS tagging: The NER uses natural language processing module to calculate all the elements in the data in terms of categories such as location, Organization, Date and time, name of a person, monetary terms, events, etc. Further, the Part of speech tagging is used to represent the elements of data in terms of nouns, pronouns, verbs, adjectives, symbols, numerals etc.

# (Step 2) Sentiment Analysis of Each Tweet: This step involves the sentiment analysis of each tweet using VADER (Valence Aware Dictionary for sEntiment Reasoning), that represents the sentiment in terms of positive, negative, neutral as well as the compound score for each Tweet. The compound score is the normalized value calculated by sum of positive and negative weights of all lexicons in the tweet.

# (Step 2. A) Sentiment Analysis for Each Word: This step includes the sentiment analysis of each word in the dataset. Further, the output from (Step 1. A) under TF-IDF is the input for (Step 2.A), where each unique term is analyzed by calculating the compound score as well as the sentiment under negative, positive or neutral category.

# (Step 2. B) Sentiment Analysis for Each Topic: This step carries sentiment analysis on each topic present in the dataset. Further, the output from (Step 1. B) under Topic Modeling is the input for (Step 2.B), where each topic is analyzed by calculating the compound score as well as the sentiment under negative, positive or neutral category.

# (Step 2. C) Sentiment Analysis on Entities: This step involves sentiment analysis on entities present in the dataset. Further, the output from (Step 1. C) under NER is the input for (Step 2.C), where each entity is analyzed by calculating the compound score as well as the sentiment under negative, positive or neutral category.

# (Step 3. A) Sentiment Analysis for a Given Topic and Named Entity: This step involves the sentiment analysis for all named entities in a topic. The input for this step is calculated by using NER on the outcomes from Topic Modeling under (Step 1.B). Further, these outcomes are analyzed by calculating the compound score as well as the sentiment under negative, positive or neutral category.

# (Step 3. B) Sentiment Analysis for a Given Topic and Word: This step involves the sentiment analysis of each word represented under topics in (Step 1.B). Further, all the words under each topic are analyzed by calculating the compound score as well as the sentiment under negative, positive or neutral category.

# A picture containing screenshot Description automatically generated

**Illustration 2 showing matrix operations under Use-Case**

# The Illustration 2 shows the detailed steps involved after the extensive data cleaning under the Demo architecture shown in Illustration 1. The Illustration 2 will help to understand the matrix operations performed as well as the workflow of other background processes discussed in the further steps.

# (Step 4) Sentiment Analysis on Topic Modeling With (NER + POS Tagging): This step involves the Topic Modeling performed on the outputs collected from NER as well as POS tagging of the dataset. This will generate a matrix with the rows as topics and the columns as elements from (NER + POS tagging) with outcomes of sentiment analysis score in each cell. Further, this matrix will be processed as input under (Step 4.A)

# (Step 4. A) Topic Modeling Matrix With (NER + POS Tagging) : This step involves taking average of the columns for the matrix outcome under (Step 4). The resulting matrix will be having only one column which holds the average of all the sentiment scores for elements under (NER + POS tagging). Nevertheless, the transpose of the resulting matrix is obtained to change the order of rows and columns.

# (Step 4. B) TF-IDF Matrix: This step involves the outcomes from (Step 1.A) stored under a sparse matrix.

# (Step 4. C) Glove Embeddings Matrix: This step includes generating a matrix for each tweet by matching each element of the tweet with the pre-trained GloVe embeddings into 50-dimensional vector space possible outcomes.

# (Step 4. D) Multiplying Glove and TF-IDF Matrix: This step carries the multiplication of outcomes from (Step 4.B) and (Step 4.C). Further, the average of columns of the resulting matrix will be taken to generate a matrix with one column.

# (Step 4. E) Final Matrix Combined from Step (4.A And 4.D): This step involves combining outcomes from (Step 4.A) and (Step 4.D). The dot product of Matrices generated by Glove embeddings and Sentiment Analysis on Topic Modeling with (NER + POS Tagging) are carried to obtain a final matrix. This final matrix will be processed further under the classification models.

# (Step 5) Classification using Machine learning Models validated by 10 CV scores:

# The final matrix is then split into training and testing data which is further operated under each of the following classification models (discussed in detail under section 5.4, chapter 5). In addition to it, the computation of algorithms from the confusion matrix (discussed in detail under section 5.5, chapter 5) are operated using the pseudo code step (#printing results) for each algorithm. However, the pseudo code for each algorithm applied under the Use-Case model is as following:

# Random Forest : The pseudo code required to implement the random forest model( Table 2).

|  |
| --- |
| 1. #Defining the function   r\_f=RandomForestClassifier ()   1. #Runtime complexity   start\_time=time. time ()   1. #Fitting the model for training   r\_f.fit (X\_train, Y\_train)   1. #Predicting the outcome   value\_pred\_r\_f = r\_f. predict (X\_test)   1. #Applying K-fold cross validation   scores\_r\_f=cross\_val\_score (r\_f, X\_test, Y\_test, cv=10)   1. #Scaling time complexity   print ("Finished in ... “, time. time()-start\_time)   1. #Printing confusion matrix   print ("Confusion Matrix obtained for random forest is \n”, confusion\_matrix (Y\_test, value\_pred\_r\_f))  8. #Printing results  print ("Accuracy after applying random forest”, (accuracy\_score (Y\_test, value\_pred\_r\_f)))  print ("Scores after applying 10 CV on random forest”, np. average(scores\_r\_f))  print ("Precision for random forest “, precision\_score(Y\_test,value\_pred\_r\_f))  print("Recall for random forest ",recall\_score(Y\_test,value\_pred\_r\_f)) |

**Table 2 Showing Pseudo Code for Random Forest Algorithm**

# Support Vector Machine Classifier: The pseudo code required to implement the Support Vector Machine Classifier( Table 3).

|  |
| --- |
| 1. #Defining the function   load\_svm=svm.SVC(C=1.0, kernel='rbf', degree=3)   1. #Runtime complexity   start\_time=time.time()   1. #Fitting the model for training   load\_svm.fit(X\_train,Y\_train)   1. #Predicting the outcome   value\_pred\_svm = load\_svm.predict(X\_test)   1. #Applying K-fold cross validation   scores\_svm=cross\_val\_score(load\_svm, X\_test,Y\_test, cv=10)   1. #Scaling time complexity   print("Finished in ... ",time.time()-start\_time)   1. #Printing confusion matrix   print("Confusion Matrix \n",confusion\_matrix(Y\_test,value\_pred\_svm))   1. #Printing results   print("Accuracy of SVM. ",(accuracy\_score(Y\_test,value\_pred\_svm)))  print("Scores after applying 10 CV on Support Vector Machine",np.average(scores\_svm))  print("Precision for Support Vector Machine" ,precision\_score(Y\_test,value\_pred\_svm))  print("Recall for Support Vector Machine ",recall\_score(Y\_test,value\_pred\_svm)) |

# Table 3 Showing Pseudo Code for SVM

# Logistic Regression: The pseudo code required to implement Logistic regression( Table 4).

|  |
| --- |
| 1. #Defining the function   load\_log\_reg=LogisticRegression(penalty='l2',random\_state=None)   1. #Runtime complexity   start\_time=time.time()   1. #Fitting the model for training   load\_log\_reg.fit(X\_train,Y\_train)   1. #Predicting the outcome   value\_pred\_log = load\_log\_reg.predict(X\_test)   1. #Applying K-fold cross validation   scores\_log\_reg=cross\_val\_score(load\_log\_reg, X\_test,Y\_test, cv=10)   1. #Scaling time complexity   print("Finished in ... ",time.time()-start\_time)   1. #Printing confusion matrix   print("Confusion Matrix \n",confusion\_matrix(Y\_test,value\_pred\_log))   1. #Printing results   print("Accuracy of Logistic Regression is",(accuracy\_score(Y\_test,value\_pred\_log)))  print("Scores after applying 10 CV on Logistic Regression",np.average(scores\_log\_reg))  print("Precision for Logistic Regression ",precision\_score(Y\_test,value\_pred\_log))  print("Recall for Logistic Regression ",recall\_score(Y\_test,value\_pred\_log)) |

# Table 4 Showing Pseudo Code for Logistic Regression Algorithm

# XGBoost with random forest as weak learner: The pseudo code for implementation of XGBoost using random forest as the base estimator also called weak learner ( Table 5).

|  |
| --- |
| 1. #Defining the function with random forest as weak learner   load\_xg\_boost = xgb.XGBRegressor(gamma=0.1,base\_estimator= r\_f, objective='binary:logistic', learning\_rate=0.01, max\_depth=10, n\_estimators=200, random\_state=1)   1. #Fitting the model for training   load\_xg\_boost.fit(X\_train, Y\_train)   1. #Predicting the outcome   val\_pred\_xg\_boost = load\_xg\_boost.predict(X\_test)   1. #Applying K-fold cross validation   scores\_xg\_boost=cross\_val\_score(load\_xg\_boost, X\_test,Y\_test, cv=10)   1. #Scaling Round off value   xg\_boost\_predictions = [round(value) for value in val\_pred\_xg\_boost]   1. #Printing confusion matrix   print("Confusion Matrix \n",confusion\_matrix(Y\_test,xg\_boost\_predictions))   1. #Printing results   print("Accuracy after applying XG Boost",(accuracy\_score(Y\_test,xg\_boost\_predictions)))  print("Scores after applying 10 CV on XG Boost",np.average(scores\_xg\_boost))  print("Precision for XG Boost ",precision\_score(Y\_test,xg\_boost\_predictions))  print("Recall for XG Boost ",recall\_score(Y\_test,xg\_boost\_predictions)) |

# Table 5 Showing Pseudo Code for XGBoost Algorithm

# AdaBoost with default decision trees as weak learner: The pseudo code for implementation of AdaBoost using default decision trees as the base estimator ( Table 6).

|  |
| --- |
| 1. #Defining the function with decision trees as default weak learner   load\_adb = AdaBoostClassifier   1. #Fitting the model for training   load\_adb.fit(X\_train,Y\_train)   1. #Predicting the outcome   val\_pred\_adb = load\_adb.predict(X\_test)   1. #Applying K-fold cross validation   scores\_adb\_=cross\_val\_score(load\_adb, X\_test,Y\_test, cv=10)   1. #Round off value   adb\_predictions = [round(value) for value in val\_pred\_adb]   1. #Printing confusion matrix   print("Confusion Matrix obtained for AdaBoost is \n",confusion\_matrix(Y\_test,adb\_predictions))   1. #Printing results   print("Accuracy after applying AdaBoost",(accuracy\_score(Y\_test,adb\_predictions)))  print("Scores after applying 10 CV on AdaBoost",np.average(scores\_adb\_))  print("Precision for AdaBoost ",precision\_score(Y\_test,adb\_predictions))  print("Recall for AdaBoost ",recall\_score(Y\_test,adb\_predictions)) |

# Table 6 Showing Pseudo Code for AdaBoost Algorithm

# Gradient Boosting with default decision trees as weak learner: The pseudo code (Table 7) represents the implementation of Gradient Boosting with default decision trees as the base estimator (weak learner).

|  |
| --- |
| * 1. #Defining the function with decision tree as default weak learner   load\_gdb = GradientBoostingClassifier(n\_estimators=100)   * 1. #Runtime complexity   start\_time=time.time()   * 1. #Fitting the model for training normalized features   load\_gdb.fit(X\_train,Y\_train)   * 1. #Predicting the outcome   value\_pred\_gdb = load\_gdb.predict(X\_test)   * 1. #Applying K-fold cross validation   scores\_gdb\_=cross\_val\_score(load\_gdb, X\_test,Y\_test, cv=10)   * 1. #Scaling Round off value   gdb\_predictions = [round(value) for value in value\_pred\_gdb]   * 1. #Compute time complexity   print("Finished in ... ",time.time()-start\_time)   * 1. #Printing confusion matrix   print("Confusion Matrix obtained for Gradient Boosting is \n",confusion\_matrix(Y\_test,gdb\_predictions))   * 1. #Printing results   print("Accuracy after applying Gradient Boosting",(accuracy\_score(Y\_test,gdb\_predictions)))  print("Scores after applying 10 CV on Gradient Boosting",np.average(scores\_gdb\_))  print("Precision for Gradient Boosting ",precision\_score(Y\_test,adb\_predictions))  print("Recall for Gradient Boosting ",recall\_score(Y\_test,adb\_predictions)) |

# Table 7 Showing Pseudo Code for Gradient Boosting Algorithm

# ROC Curve for all algorithms: The Pseudo code for plotting ROC curve to calculate area under the curve is shown under table 8

|  |
| --- |
| plt.figure(0).clf() #creating figureplt.plot(fpr\_rf, tpr\_rf, '', label = 'AUC = %0.2f Random Forest' % roc\_auc\_rf)plt.plot(fpr\_svm, tpr\_svm, '', label = 'AUC = %0.2f SVM' % roc\_auc\_svm)plt.plot(fpr\_log, tpr\_log, '', label = 'AUC = %0.2f Logistic Regression' % roc\_auc\_log\_reg)plt.plot(fpr\_xg\_boost, tpr\_xg\_boost, '', label = 'AUC = %0.2f XG Boost' % roc\_auc\_xg\_boost)plt.plot(fpr\_adb, tpr\_adb, '', label = 'AUC = %0.2f AdaBoost' % roc\_auc\_adb)plt.plot(fpr\_gdb, tpr\_gdb, '', label = 'AUC = %0.2f Gradient Boost' % roc\_auc\_gdb)plt.plot()plt.legend(loc=4) |

**Table 8 Showing Pseudo Code for Bi-Lstm Algorithm**

# 8. Bi-LSTM Model: The pseudo code for implementation of bidirectional LSTM (Bi-LSTM) is represented under table 9

**Table 9 Showing Pseudo Code for Bi-Lstm Algorithm**

|  |
| --- |
| 1. #creating check of maximum words   MAX\_NUM\_WORDS\_check = len(counts)   1. #tokenizing context   tokenizer\_check= Tokenizer(num\_words=MAX\_NUM\_WORDS\_check)  tokenizer\_check.fit\_on\_texts(cleaned\_tweets)  word\_vector\_obtained = tokenizer\_check.texts\_to\_sequences(cleaned\_tweets)  word\_index\_obtained = tokenizer\_check.word\_index   1. #checking size of vocab   vocab\_size\_check = len(word\_index\_obtained)   1. #printing vocab for display   vocab\_size\_check   1. #defining sequence length   MAX\_SEQ\_LENGTH\_CHECK = 10   1. #creating input tensor   input\_tensor\_obtained = pad\_sequences(word\_vector\_obtained, maxlen=MAX\_SEQ\_LENGTH\_CHECK)  print(input\_tensor\_obtained.shape)   1. #defining shape of embeddings   EMBEDDING\_DIMENSIONS = 50   1. #creating embedding matrix   embedding\_matrix = np.zeros((MAX\_NUM\_WORDS\_check, EMBEDDING\_DIMENSIONS))   1. #initializing neural nets   input\_obtained = Input(shape=(MAX\_SEQ\_LENGTH\_CHECK,))   1. #Embedding\_Dimensions   x\_obtained = Embedding(MAX\_NUM\_WORDS\_check, weights=[word\_matrix\_embeddings])(input\_obtained)  x\_obtained = Bidirectional(LSTM(10 , recurrent\_dropout=0.01,return\_sequences=True,dropout=0.20))(x\_obtained)   1. #preparing inputs   x\_obtained = GlobalMaxPool1D()(x\_obtained)   1. #applying relu effect   x\_obtained = Dense(10, activation=”relu”)(x\_obtained)  x\_obtained = Dropout(0.20)(x\_obtained)   1. #applying sigmoid reg   x\_obtained = Dense(1, activation=”sigmoid”)(x\_obtained)   1. #fitting neural nets on data   model\_lstm.fit(x\_train, y\_train, batch\_size=8, epochs=5) |

# Long Short-Term Memory: The pseudo code for implementation of LSTM is represented under table 10

**Table 10 Showing Pseudo Code for LSTM**

|  |
| --- |
| 1. #loading lstm for sequential classification   model\_lstm\_load = Sequential()   1. # Embedding\_Dimensions 2. model\_lstm\_load.add(Embedding(len(word\_matrix\_embeddings),EMBEDDING\_DIMENSIONS,weights=[word\_matrix\_embeddings],input\_length=10, trainable=False)) 3. #Setting other parameters   model\_lstm\_load.add(Conv1D(kernel\_size=3, filters=32, activation='relu',padding='same'))   1. #adding pool size   model\_lstm\_load.add(MaxPooling1D(pool\_size=2))   1. #selecting parameter for dropout   model\_lstm\_load.add(Dropout(0.2))   1. #add layer   model\_lstm\_load.add(LSTM(300))   1. #adding sigmoid parameter   model\_lstm\_load.add(Dense(1, activation='sigmoid'))   1. #defining number of epochs   EPOCHS=5   1. #defining early stop   early\_stop = EarlyStopping(monitor='val\_loss', patience=3)   1. #Fitting and training the model   model\_seq\_hist = model\_lstm\_load.fit(x\_train, y\_train,epochs=EPOCHS, batch\_size=12, shuffle=True,callbacks=[early\_stop]) |

**1. 4 System model for Tweet Level**

The system model will explain the steps involved in transforming tweets into a useful set of information to predict depression in users on social media (Twitter). The tweets will be converted into a matrix using various operations discussed further in this section. The steps involved are as following:

1. Let T be the set of n tweets, where (T) = { t1, t2, t3, ...tn }
2. Let each tweet TK has m words represented by (w) where (TK) = { w1, w2, w3, ....wm}
3. Now, TF-IDF Matrix of all Tweets will be = (T \* TK) = { t1, t2, t3, ...tn } \* { w1, w2, w3 ,....wm}
4. Let D be the dimensions of GloVe Embeddings = (d)
5. Now, the Glove Matrix for all Tweets will be = (TK \* d) = { w1, w2, w3, ...wm} \* (d)
6. Let GT be the matrix generated by multiplying the matrix for TF-IDF (step 3) and GloVe (step 5) we get (GT)= (T \* TK) (TK \*d) = (T \* d) = { t1, t2, t3, ...tn } \* (d)
7. Let (TOP) be the matrix generated after the application of sentiment analysis on topic modeling of words obtained by (NER + POS) on (T), where NER is Named Entity Recognition and POS is part of speech tagging
8. Now, if we take (X) number of topics and (Y) number of passes, the TOP matrix will be (TOP) = (X \* Y)
9. In order to multiply the GT and TOP matrix, the columns of one matrix should be equal to rows of other matrix which can be achieved using next steps.
10. Let GT\_AVG be the matrix generated by taking average of column values of GT matrix(step 6), we get GT\_AVG = { t1, t2, t3, ..tn } \* 1
11. Let TOP\_AVG be the matrix generated by taking average of column values of TOP matrix(step 8), we get TOP\_AVG = (X \* 1)
12. Let TOP\_AVG\_T be the matrix generated by taking transpose of the TOP\_AVG matrix (step 11), we get TOP\_AVG\_T=(1 \* X)
13. Let FM be the Final Matrix generated by taking dot product of (GT\_AVG) and (TOP\_AVG\_T), we get FM = ({ t1, t2, t3, ..tn } \* 1 ) (1 \* X) = { t1, t2, t3, ..tn } \* X
14. The Final matrix for classification under machine learning models will be FM = { t1, t2, t3, ...tn } \* (X)

# Moreover, to understand the steps involved under the Use-Case Model, let’s take an example to discuss statistical operations involved in generating a matrix from tweets for further processing under classification models. The steps to obtain a matrix for classification models involved in this Use-Case are as following:

1. Let (t) be the number of tweets in this case = 500
2. Let (v) be the length of vocabulary(total words in dataset) in this case = 640
3. Let g = dimensions of glove embeddings, in this case = 50
4. Let GT be the output from (matrix of TF-IDF \* glove embeddings matrix ) = ( t \* v) ( v \* g) = (500 \* 640) (640 \* 50) = (500 \* 50)
5. Let TOP be the output of matrix obtained after sentiment analysis(compound values) on Topic modeling results with NER + POS tagging with dimensions topics (dt) = 10, passes (p) = 10 so the dimensions of TOP will be = (dt \* p) = 10 \* 10
6. In order to multiply GT and TOP, the columns of 1st matrix should be equal to rows of 2nd matrix which can be achieved by performing next steps.
7. Average column values of GT are obtained and reduced to 500 \* 1, Let’s say GT\_AVG
8. Average column values of TOP are obtained and reduced to 10 \* 1, Let’s say TOP\_AVG
9. Now take transpose of TOP\_AVG, which will result in dimensions of 1 \* 10, Let’s say TOP\_AVG\_T
10. Now multiply GT\_AVG and TOP\_AVG\_T, resulting into dimensions, (500 \* 1) (1 \* 10) = (500 \* 10)
11. The Final Matrix let’s say FM will be the output from Step 10, FM = (500\* 10). This Final matrix will be operated as input under various classification models.

Nevertheless, Let’s take an example of a single tweet (T1) to represent the matrix output for the Use-Case demo model. The following results represent the steps of matrix operations for a single

tweet that will be finally processed under classification models.

Step 1. **Tf-Idf Matrix**: Illustration 3 shows the matrix for a single tweet (T1) obtained by using TF-IDF.

A screenshot of a cell phone

Description automatically generated

**Illustration 3 Showing Tf-Idf matrix for a single Tweet**

Step 2. **GloVe Matrix**: Illustration 4 shows a 50-dimension matrix for a single tweet (T1) obtained by pre-trained GloVe embeddings.

A screenshot of a cell phone

Description automatically generated

**Illustration 4 Showing GloVe Embedding matrix for a single Tweet**

Step 3. **TF-IDF + GloVe Matrix**: Illustration 5 shows the matrix for a single tweet (T1) obtained by combining the outputs of TF-IDF and Glove Embeddings (illustration 3 and 4).

A close up of a newspaper

Description automatically generated

**Illustration 5 Showing Tf-Idf and GloVe matrix for a single Tweet**

Step 4. **Mean Matrix of TF-IDF + GloVe**: Illustration 6 represents the outcome for a single tweet (T1) obtained by taking average of matrix resulted from combination of TF-IDF and GloVe ( Illustration 5). However, by taking the average, the resulting matrix gets reduced to a single dimension with 1 column.

A picture containing table

Description automatically generated

**Illustration 6 Showing Average of Tf-Idf and GloVe matrix for a single Tweet**

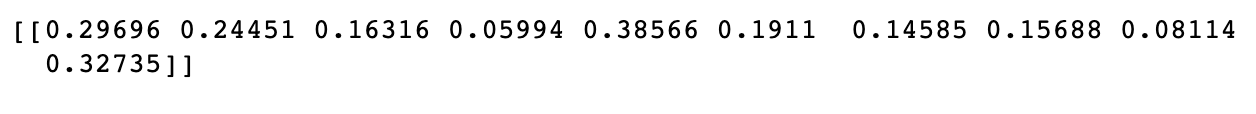
Step 5. **Matrix Generated by** **Sentiment Analysis on Topic Modeling (NER + POS Tagging)**: Illustration 7 represents the matrix obtained by sentiment analysis on topic modeling of the terms, that are generated by performing NER + POS tagging on the dataset.

A close up of a piece of paper

Description automatically generated

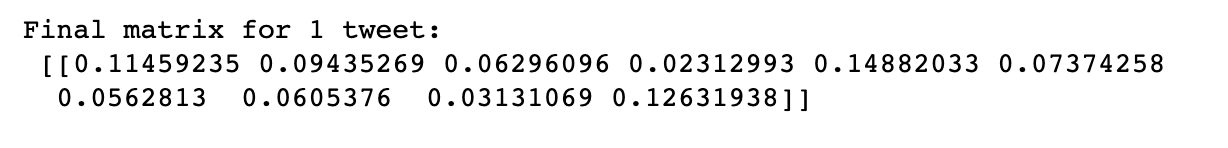
**Illustration 7 Showing matrix from sentiment analysis on Topic Modeling using NER and POS tagging**

Step 6. **Matrix Obtained by taking Average and Transpose of the matrix generated in Step 5**: In order to multiply two matrices, the columns of the first matrix should be equal to the rows of another matrix. However, by taking average of the matrix in step 5, the resulting matrix becomes (10 \* 1) which cannot be combined by the matrix in step 4 with a single dimension. Therefore, by taking the transpose, the matrix converts into (1 \* 10) as shown in Illustration 8.



**Illustration 8 Showing transpose for matrix obtained by average matrix of sentiment analysis on Topic Modeling (NER + POS tagging)**

Step 7. **Final Matrix**: Illustration 9 shows the final matrix generated by the dot product of matrices obtained under step 4 and step 6. However, the dimensions of the final matrix for the single tweet (T1) are represented as (1\*10). This matrix will be further processed as input for the classifier models.



**Illustration 9 Showing Final matrix for a single Tweet**