National Institute of Technology Patna

MINOR PROJECT (Group -17)

CONTEXT BASED SENTIMENT ANALYSIS
WITH SARCASM DETECTION ON TWITTER TWEETS

Submitted By:-

Amarjeet Chaudhary – 1806116
Swapna Sahani – 1806134
Rohit Singh – 1806005

In the Guidance of Dr. Anubha Maurya

Introduction :

Problem Statement : Context based Sentiment Analysis of tweets and retweets with Sarcasm

This Presentation describes the behavior of data for sarcastic , topic from which it belongs to and its sentiment , with the help of CNN,LSTM,NMF,LR and Cosine-Similarity.

We will be able to understand the basic architecture of above mentioned standard algorithm and technique that is being used.

Project Motivation:

- We can use Machine Learning in Finance, Medicine, almost everywhere. That’s why we decided to conduct our project around the Machine Learning

- As it is the most hot topic where a machine is learning the ability to learn the human nature of talking and thinking. A model is build , that uses the same and implement in real world.

- This project was motivated by our desire to investigate the complete description of the tweets

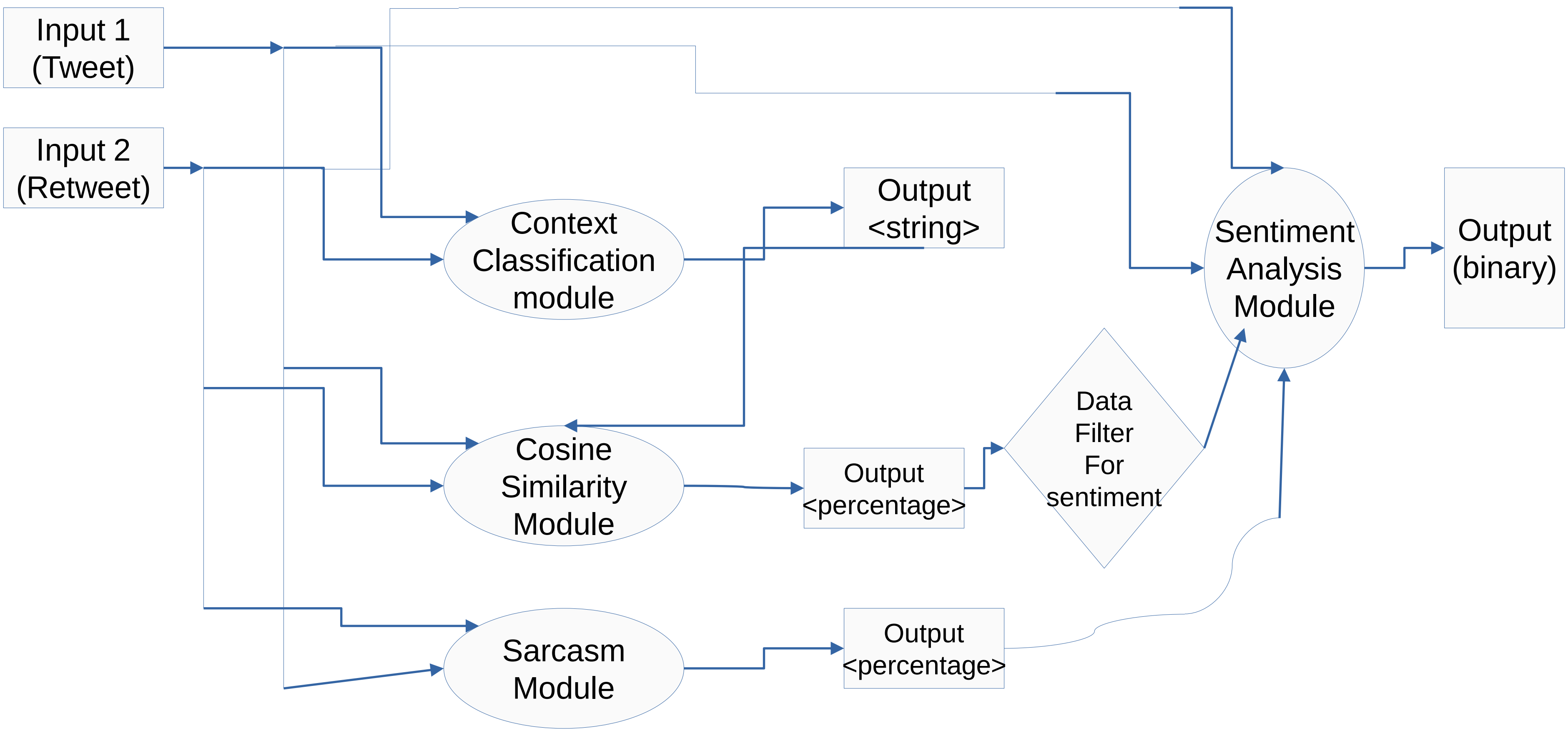
Work performed :

- Sarcasm Detection
- Context Classification
- Sentiment Analysis
- Cosine Similarity
- Semantic Similarity using Spacy model

Procedure :

- Logistic Regression(LR)
- Convolutional Neural Network (CNN)
- Non-negative Matrix Factorization (NMF)
- Long Short Term Memory (LSTM)
- TfidfVectorizer
- Cosine Similarity

Data Flow Diagram :



Description of Term Used in Data Flow Diagram :

Input1 and Input2:
These are the tweet and retweet respectively which will be used as input for the module.

ContextClassificationModule:
This will take Input1 and Input2 as input and gives output as a string that classifies the Input1 and Input2 according to their context.

SarcasmModule:
This will take Input1 and Input2 as input and gives output as the percentage of sarcastic behavior of input.

CosineSimilarityModule:
It will take Input1 , Input2 and output from ContextClassificationModule which is used to calculate the similarity percentage between two inputs which will be further used to filter out the necessary data.

SentimentAnalysisModule:
It will take the filtered data and output from SarcasmModule to calculate the sentiment for the input,output will be in the vector of binary form.

Algorithm :

```
Def comp(s, threshold, value):
        # if sarcastic_value > threshold return false to reverse the polarity else return true for no change
        # value used here  is the sarcastic value obtained using SarcasmModule()
        if(value > threshold):
                return false
        else:
                return true
Def model(s1, s2):
        data = [s1, s2]
        # STEP 1  find the topic for the data
        topic_decided = ContextClassificationModule(s1, s2)

        # STEP 2 calculate the similarity between data
        similarity_percentage = CosineSimilarityModule(s1, s2, topic_decided)

        # STEP 3 calculate the sarcastic percentage for each data
        sarcasm_percentage = SarcasmModule(s1, s2)

        # STEP 4 calculate sentiment using previous output as input along with comp() function for reverse the output where needed
        final_output = SentimentAnalysisModule(s1, s2, sarcasm_percentage, comp)
```

NOTE: ContextClassificationModule() , CosineSimilarityModule() , SarcasmModule() , SentimentAnalysisModule() are the modules created and takes input as maintained

Datasets:

Dataset-I for Context classification:

We have created a dataset of 5434 sentences from Wikipedia of different Topics such as 'sports', 'politics' and 'terrorism'.

There are approximate equal sentences of each topic (approximately 1800 from each)

Dataset – II for Sarcasm Detection:

We have a data set of 26709 tweets in English coming from Kaggle.

It is composed of 2 columns that are headline and class.

class contains a binary value, 0 if the tweet is not sarcasm,

1 if the tweet is sarcasm.

Dataset-III for Sentiment Analysis:

We have a dataset of 20000 tweets in English coming from the bz format file where 9743 negative and 10257 positive.

It contains paragraphs for positive tweets and negative tweets and 0 if tweet is negative and 1 if tweet is positive.

	data	category
4980	It is alleged that the Qatar Charity gave fina...	terrorism
137	Plays, farces, spectacles, gladiators, strang...	sports
2491	In the United States, elections for public of...	politics
4533	In view of these considerations, violence may...	terrorism
542	This is the first description of a "kicking g...	sports

Dataset-I

Headline	is_sarcastic
former versace store clerk sues over secret 'b...	0
the 'roseanne' revival catches up to our thorn...	0
mom starting to fear son's web series closest ...	1
boehner just wants wife to listen, not come up...	1
j.k. rowling wishes snape happy birthday in th...	0

Dataset-II

```
Out[17]: ['stuning even for the non gamer  this sound track was beautiful  it paints the senery in your mind so well i would recomend
         it even to people who hate vid  game music  i have played the game chrono cross but out of all of the games i have ever playe
         d it has the best music  it backs away from crude keyboarding and takes a fresher step with grate guitars and soulful orchest
         ras  it would impress anyone who cares to listen    ',
          'the best soundtrack ever to anything   i m reading a lot of reviews saying that this is the best  game soundtrack  and i fi
         gured that i d write a review to disagree a bit  this in my opinino is yasunori mitsuda s ultimate masterpiece  the music is
         timeless and i m been listening to it for years now and its beauty simply refuses to fade the price tag on this is pretty sta
         ggering i must say  but if you are going to buy any cd for this much money  this is the only one that i feel would be worth e
         very penny ',
          'amazing   this soundtrack is my favorite music of all time  hands down  the intense sadness of  prisoners of fate   which m
         eans all the more if you ve played the game  and the hope in  a distant promise  and  girl who stole the star  have been an i
         mportant inspiration to me personally throughout my teen years  the higher energy tracks like  chrono cross   time s scar
         time of the dreamwatch   and  chronomantique   indefinably remeniscent of chrono trigger  are all absolutely superb as well t
         his soundtrack is amazing music  probably the best of this composer s work  i haven t heard the xenogears soundtrack  so i ca
         n t say for sure   and even if you ve never played the game  it would be worth twice the price to buy it i wish i could give
         it  stars ',
          'excellent soundtrack  i truly like this soundtrack and i enjoy video game music  i have played this game and most of the mu
         sic on here i enjoy and it s truly relaxing and peaceful on disk one  my favorites are scars of time  between life and death
```

Dataset-III

1. Data Generation for Context Classification :

```
In [1]: import pandas as pd
        import wikipedia as wp
```

```
In [3]: sports_list = ["sport","cricket","football","basketball","wrestling","kabaddi","hockey","Game"]
        politics_list = ["Politics","Politician","Democratic republic","Legislature","Parliament","Election","Government","President (gov
        entertainment_list =["Entertainment","music","song","movie","Comedy","film","circus"]
        education_list = ["Education","Mathematics","Chemistry","Course (education)","Student","Test (assessment)","Teacher"]
        terrorism_list = ["Terrorism","War","Violence","Jihad","Al-Qaeda"]
```

```
In [4]: def get_data(parameter):
            var_data = []
            for i in parameter:
                data1 = wp.page(i)

                s = ""
                for character in data1.content:
                    if character != '(' or character != ')' :
                        s = s + character
                    if(character == '.' or character == '\n'):
                        if (len(s) >= 60):
                            var_data.append(s)
                        s = ""
            return var_data
```

```
In [5]: list1 = get_data(sports_list)
```

```
In [6]: list1
```

```
Out[6]: ['Sport pertains to any form of competitive physical activity or game that aims to use, maintain or improve physical ability
        and skills while providing enjoyment to participants and, in some cases, entertainment to spectators.',
         " Sports can, through casual or organized participation, improve one's physical health.",
         ' Hundreds of sports exist, from those between single contestants, through to those with hundreds of simultaneous participan
```

[fig 1.1]   Above fig. Show the creation of data using Wikipedia module in python

Converting To Csv :

```
In [16]: df1 = pd.DataFrame(list1)
         df1.to_csv("1sports_csv.csv",index=False)
         df2 = pd.DataFrame(list2)
         df2.to_csv("2enter_csv.csv",index=False)
         df3 = pd.DataFrame(list3)
         df3.to_csv("3education_csv.csv",index=False)
         df4 = pd.DataFrame(list4)
         df4.to_csv("4politics_csv.csv",index=False)
         df5 = pd.DataFrame(list5)
         df5.to_csv("5terrorism_csv.csv",index=False)
```

```
In [17]: data1 = pd.read_csv("1sports_csv.csv")
         data1["category"] = "sports"
         data1.rename(columns={'0': 'data'}, inplace=True)
         data1
```

Out[17]:

	data	category
0	Sport pertains to any form of competitive phys...	sports
1	Sports can, through casual or organized parti...	sports
2	Hundreds of sports exist, from those between ...	sports
3	In certain sports such as racing, many contes...	sports
4	Some sports allow a "tie" or "draw", in which...	sports

[fig 1.2] Converting the data to csv format for further use

Creating Context classification Module with NMF:

- Non-Negative Matrix Factorization (NMF) is an unsupervised technique so there are no labeling of topics that the module will be trained on. The way it works is that, NMF decomposes (or factorizes) high-dimensional vectors into a lower-dimensional representation. These lower-dimensional vectors are non-negative which also means their coefficients are non-negative.
- Using the original matrix (A), NMF will give you two matrices (W and H). W is the topics it found and H is the coefficients (weights) for those topics. In other words, A is articles by words (original), H is articles by topics and W is topics by words.
- So assuming 301 articles, 5000 words and 30 topics we would get the following 3 matrices:

```
A = tfidf_vectorizer.transform(texts)
W = nmf.components_
H = nmf.transform(A)

A = 301 x 5000
W = 30 x 5000
H = 301 x 30
```

Processing with TfidfVectorizer :

- TfidfVectorizer(max_df=max_df,min_df=min_df,stop_words='english')
- max_df is used for removing data values that appear too frequently, also known as "corpus-specific stop words".
- min_df is used for removing terms that appear too infrequently.
- **Example :**
- max_df = 0.90 means "It ignores terms that appear in more than 90% of the documents".
- max_df = 90 means "It ignores terms that appear in more than 90 documents".
- min_df = 0.02 means "ignore terms that appear in less than 2% of the documents".
- min_df = 2 means "ignore terms that appear in less than 2 documents".

Word to Real Number Output :

```
In [9]: from sklearn.feature_extraction.text import TfidfVectorizer
        tfidf = TfidfVectorizer(max_df=0.9,min_df=2,stop_words='english')
        tfidf
```

```
In [10]: dtm = tfidf.fit_transform(df['tidy_tweet'])
         dtm
```

```
Out[10]: <5434x6091 sparse matrix of type '<class 'numpy.float64'>'
                 with 54894 stored elements in Compressed Sparse Row format>
```

```
In [15]: print(dtm)
```

```
  (0, 5191)	0.2717584746755791
  (0, 1915)	0.2382280567264727
  (0, 818)	0.22129622665790666
  (0, 3947)	0.22254204396656543
  (0, 1902)	0.27723115527303666
  (0, 4361)	0.2837039625774394
  (0, 5103)	0.22656545598077138
  (0, 7)	0.2526229399185298
  (0, 2721)	0.2526229399185298
  (0, 3321)	0.2497812516837208
  (0, 160)	0.22656545598077138
  (0, 2333)	0.14397674633805269
  (0, 61)	0.22129622665790666
```

[fig 1.3]

Snippet for Context Classification module component :

```
In [10]: from sklearn.decomposition import NMF
         nmf_model = NMF(n_components=3,random_state=42)

In [11]: nmf_model.fit(dtm)

Out[11]: NMF(alpha=0.0, beta_loss='frobenius', init=None, l1_ratio=0.0, max_iter=200,
             n_components=3, random_state=42, shuffle=False, solver='cd', tol=0.0001,
             verbose=0)

In [12]: for i, arr in enumerate(nmf_model.components_):
             print(f"The words of NMF of high frequency of {i} is")
             print([tfidf.get_feature_names()[i] for i in arr.argsort()[-25:]])
             print('\n')

         The words of NMF of high frequency of 0 is
         ['world', 'international', 'england', 'popular', 'league', 'australian', 'teams', 'century', 'sports', 'play', 'cricket', 'spor
         t', 'known', 'association', 'team', 'hockey', 'rugby', 'basketball', 'players', 'games', 'rules', 'ball', 'played', 'game', 'fo
         otball']


         The words of NMF of high frequency of 1 is
         ['cabinet', 'term', 'house', 'union', 'office', 'power', 'constitution', 'parliamentary', 'ministers', 'executive', 'republic
         s', 'india', 'title', 'elected', 'council', 'parliament', 'state', 'republic', 'united', 'head', 'minister', 'prime', 'governme
         nt', 'states', 'president']


         The words of NMF of high frequency of 2 is
         ['forms', 'term', 'military', 'group', 'terrorist', 'according', 'attacks', 'politics', 'international', 'policy', 'people', 'u
         sed', 'health', 'islamic', 'groups', 'social', 'public', 'world', 'state', 'governance', 'alqaeda', 'terrorism', 'jihad', 'poli
         tical', 'violence']
```

[fig 1.4]

Output :

In [21]:
```
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
print("accuracy score is ")
print(accuracy_score(df['category'],df['category_decided']))
print("classification report is ")
print(classification_report(df['category'],df['category_decided']))
print("confusion matrix is ")
print(confusion_matrix(df['category'],df['category_decided']))
```

```
accuracy score is
0.7926120382523908
classification report is
              precision    recall  f1-score   support

    politics       0.83      0.61      0.70      1700
      sports       0.96      0.84      0.89      1784
   terrorism       0.67      0.92      0.78      1849

    accuracy                           0.79      5333
   macro avg       0.82      0.79      0.79      5333
weighted avg       0.82      0.79      0.79      5333

confusion matrix is
[[1035   43  622]
 [  82 1491  211]
 [ 128   20 1701]]
```

[fig 1.5]   Output shows the classification report of context classification module

2. Sarcasm Detection Module With CNN:

- A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.
- The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

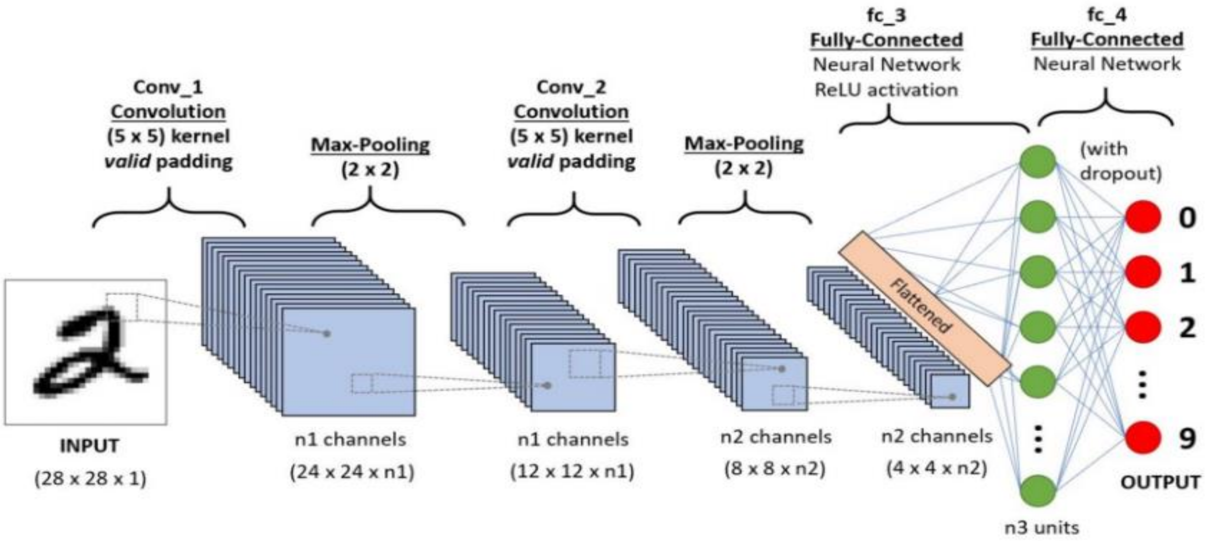
Working of CNN :

35	19	25	6
13	22	16	53
4	3	7	10
0	8	1	3

22 activated →

35	19	25	6
13	$\boxed{22}$	16	53
4	3	7	10
0	8	1	3

Matrix Value is updated as per the neuron weight Updation formula.



[fig 2.1 reference : towardsdatascience]

How it works for words ?

Images are just some points in space,just like

the word vectors are. By representing each word

with a vector of numbers of a specific length

and stacking a bunch of words on top

of each other, we get an “image.” Computer vision

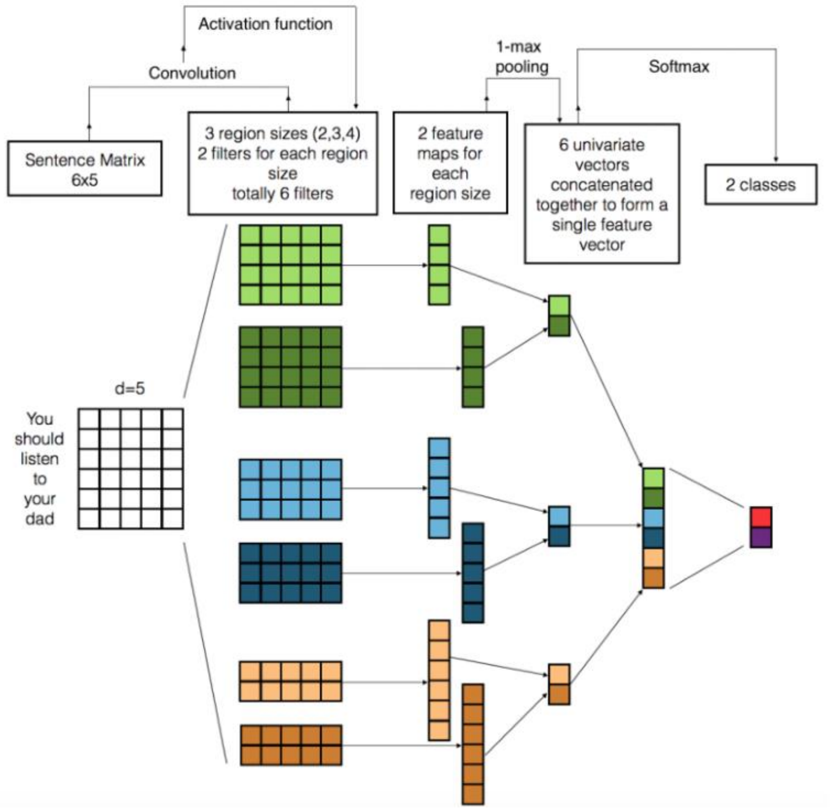
filters usually have the same width and height

and slide over local parts of an image. In NLP,

we typically use filters that slide over word

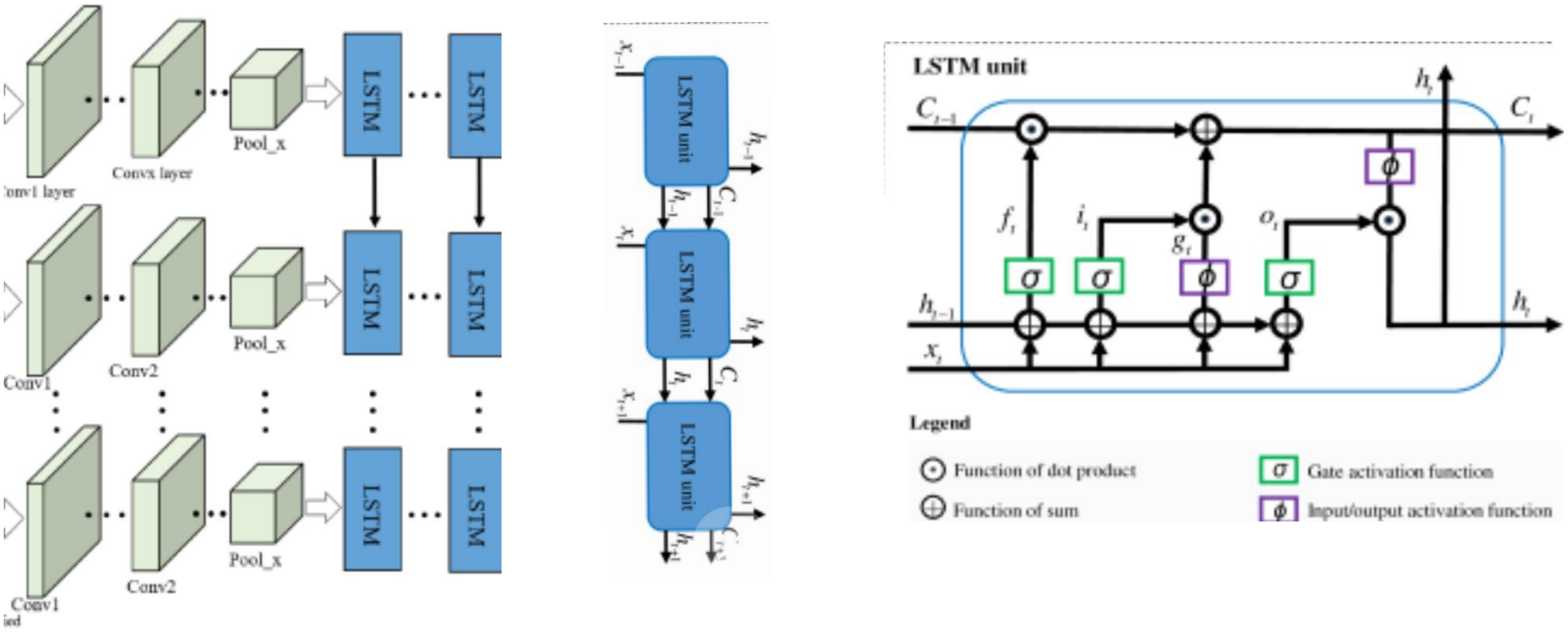
embeddings — matrix rows. Therefore, filters

usually have the same width as the length of the word embeddings.



[fig 2.2 reference : machinelearningmastery]

2.1 Adding New Layer(LSTM) in the module :



[fig 2.3 reference : quora]

Snippet of Sarcasm Detection module :

```
In [8]: tokenizer= Tokenizer(num_words=vocab_size, oov_token='OOV')
        tokenizer

Out[8]: <keras_preprocessing.text.Tokenizer at 0x2368a8cf1c8>

In [12]: tokenizer.fit_on_texts(X_train)

In [11]: training_sequences=tokenizer.texts_to_sequences(X_train)
         training_padded=pad_sequences(training_sequences, maxlen=max_len, padding=padding_type, truncating=trunc_type)

In [12]: testing_sequences=tokenizer.texts_to_sequences(X_test)
         testing_padded=pad_sequences(testing_sequences, maxlen=max_len, padding=padding_type, truncating=trunc_type)

In [13]: def create_model(vocabulary_size,embedding_dim,seq_len):
             model=Sequential()
             model.add(Embedding(vocabulary_size,embedding_dim,input_length=seq_len))
             model.add(LSTM(64,dropout=0.2,recurrent_dropout=0.25))
             model.add(Dense(1,activation='sigmoid'))
             opt = keras.optimizers.Adam(learning_rate=0.01)
             model.compile(loss='binary_crossentropy',optimizer=opt,metrics=['accuracy'])
             model.summary()
             return model
```

[fig 2.4]

2.2 Snippet of Sarcasm Detection module2 :

```
In [21]: model2 = Sequential()
         model2.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
         model2.add(Flatten())

         model2.add(Dense(units=32,activation='relu'))
         model2.add(Dropout(0.5))

         model2.add(Dense(units=10,activation='relu'))
         model2.add(Dropout(0.5))

         model2.add(Dense(units=1,activation='sigmoid'))
         opt = keras.optimizers.Adam(learning_rate=0.01)
         model2.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
         model2.summary()

         es = EarlyStopping(monitor='val_loss', mode='min', verbose=2, patience=7)
         model2.fit(training_padded,y_train,batch_size=64,epochs=15,verbose=2,validation_data=(testing_padded,y_test),callbacks=[es])

         Model: "sequential_1"
         _________________________________________________________________
```

[fig 2.5]

Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

Sarcasm Detection module Output Accuracy :

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 32, 32)            96000
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                32800
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
_________________________________________________________________
dropout_1 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 129,141
Trainable params: 129,141
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
606/606 - 1s - loss: 0.4100 - accuracy: 0.8152 - val_loss: 0.2762 - val_accuracy: 0.8862
Epoch 2/15
606/606 - 1s - loss: 0.2595 - accuracy: 0.8988 - val_loss: 0.2419 - val_accuracy: 0.9011
Epoch 3/15
606/606 - 1s - loss: 0.1834 - accuracy: 0.9295 - val_loss: 0.2361 - val_accuracy: 0.9179
Epoch 4/15
606/606 - 1s - loss: 0.1414 - accuracy: 0.9482 - val_loss: 0.2656 - val_accuracy: 0.9265
Epoch 5/15
606/606 - 1s - loss: 0.1093 - accuracy: 0.9595 - val_loss: 0.3202 - val_accuracy: 0.9247
Epoch 6/15
606/606 - 1s - loss: 0.0950 - accuracy: 0.9650 - val_loss: 0.2906 - val_accuracy: 0.9290
Epoch 7/15
606/606 - 1s - loss: 0.0844 - accuracy: 0.9683 - val_loss: 0.4429 - val_accuracy: 0.9287
Epoch 8/15
606/606 - 1s - loss: 0.0761 - accuracy: 0.9725 - val_loss: 0.4329 - val_accuracy: 0.9342
Epoch 9/15
606/606 - 1s - loss: 0.0704 - accuracy: 0.9720 - val_loss: 0.5098 - val_accuracy: 0.9360
Epoch 10/15
606/606 - 1s - loss: 0.0670 - accuracy: 0.9746 - val_loss: 0.5067 - val_accuracy: 0.9361
Epoch 00010: early stopping
```

```
[21]: <tensorflow.python.keras.callbacks.History at 0x22a92ac3f88>
```

```
In [22]: def prediction_text2(sent):
             sent=[sent]
             seq=tokenizer.texts_to_sequences(sent)
             padded=pad_sequences(seq,maxlen=max_len,padding=padding_type, truncating=trunc_type)
             return model2.predict(padded)
```

```
In [28]: sent="you broke my car , good job"
         print(prediction_text2(sent))

         [[0.99999976]]
```

[fig 2.6]

How sarcastic statement changes the sentiment:

```
In [18]: #without sarcasm
dd = pd.DataFrame([[sent[0]]],columns=['data'])
dd2 = pd.DataFrame([[sent2]],columns=['data'])
with open("count_v.pkl","rb") as f:
    vec1 = pickle.load(f)
vec = CountVectorizer(decode_error="replace", vocabulary=vec1)
for i in model3.predict(vec.transform(dd['data'])):
    if(i == 1):
        print(sent," has a positive sentiment")
    else:
        print(sent," has a negative sentiment")
for i in model3.predict(vec.transform(dd2['data'])):
    if(i == 1):
        print(sent2," has a positive sentiment")
    else:
        print(sent2," has a negative sentiment")

['you broke my car , good job']  has a positive sentiment
i am a good boy  has a positive sentiment
```

```
In [17]: #cross check for sarcasm
for i in model3.predict(vec.transform(dd['data'])):
    if(i == 1 and x[0][0] > 0.3):
        print(sent," has a negative sentiment")
    else:
        print(sent," has a positive sentiment")
for i in model3.predict(vec.transform(dd2['data'])):
    if(i == 1):
        print(sent2," has a positive sentiment")
    else:
        print(sent2," has a negative sentiment")

['you broke my car , good job']  has a negative sentiment
i am a good boy  has a positive sentiment
```

fig 0.2

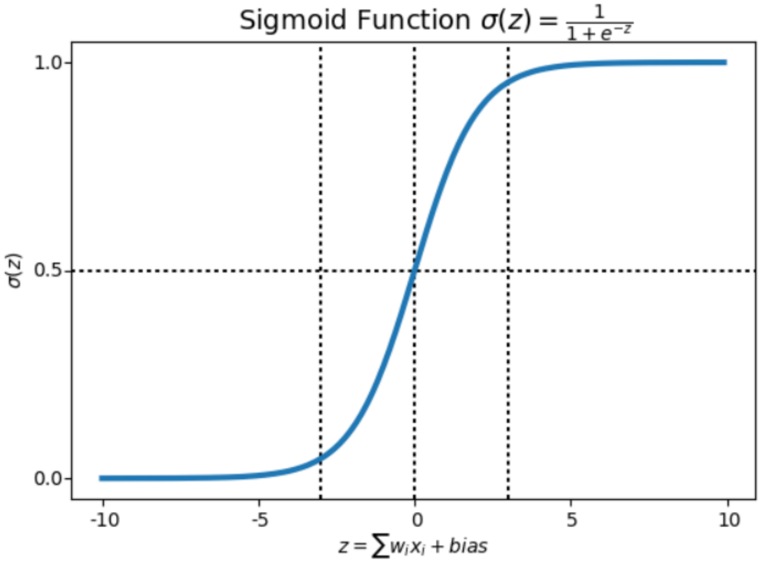
3. Sentiment Analysis Module :

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability. It is based on most likelihood parameter

We can call a Logistic Regression a Linear Regression module but the Logistic Regression uses a more complex cost function, this cost function can be defined as the ‘Sigmoid function’ or also known as the ‘logistic function’ instead of a linear function.

Sigmoid Function :

In order to map predicted values to probabilities, we use the Sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.



[Graph of sigmoid function]

Input Processing :

"Today I am happy"

↓ tokenizer

'today' , 'I' , 'am' , 'happy'

↓ Creating vocabulary from data

'among' , 'ants' , ……… , 'happy' , 'zoo'

↓ Encoding sparse matrix

'among' , 'ants' , ……… , 'happy' , 'zoo'
[0,…,0,1,0,…1,0,.....1,1,.....,1,0,…..1]

Processing bag of words

```
: sen = ["today i am happy"]

: from sklearn.feature_extraction.text import CountVectorizer
  cv = CountVectorizer(binary=True)
  import pandas as pd
  data = pd.DataFrame(sen)

: cv.fit(data.iloc[:,-1])
  x = cv.transform(data.iloc[:,-1])
  x

: <1x3 sparse matrix of type '<class 'numpy.int64'>'
        with 3 stored elements in Compressed Sparse Row format>
```

[fig 3.2]

cv.fit() - it creates a vobulary over raw
Input of the document
cv.transform() - it transform the tokens
Into sparse matrix

How Model.fit() works :

```
n [29]: x_train

ut[29]: array([[     0,      0,      0, ...,   632,    126,     23],
               [     0,      0,      0, ...,  3841,     16,   3842],
               [     0,      0,      0, ...,   754,    366,     30],
               ...,
               [     0,      0,      0, ...,   284,    406,     22],
               [     0,      0,      0, ...,  1723,     26,   1551],
               [     0,      0,      0, ..., 11305,    197,    223]])
```

y_train

```
([[0],
  [0],
  [1],
   .
   .
  [0],
  [1],
  [1]])
```

[fig 3.3]

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=c)
lr.fit(X_train, y_train)
print (accuracy_score(y_val,lr.predict(X_val)))

Where C is the inverse regularization parameter

Output Snippet for Sentiment Module :

In [47]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

In [48]:
```
X_train, X_val, y_train, y_val = train_test_split(X, train_labels, train_size = 0.75)
```

In [49]:
```
for c in [0.01, 0.05, 0.25, 0.5, 0.75, 1]:
    lr = LogisticRegression(C=c)
    lr.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"% (c, accuracy_score(y_val, lr.predict(X_val))))
```

```
Accuracy for C=0.01: 0.8476
Accuracy for C=0.05: 0.868
```

In [56]:
```
sen = input('enter string')
```

```
enter stringhe died
```

In [57]:
```
dd = pd.DataFrame([[sen]],columns=['data'])
dd
```

Out[57]:

	data
0	he died

In [58]:
```
lr.predict(cv.transform(dd['data']))
```

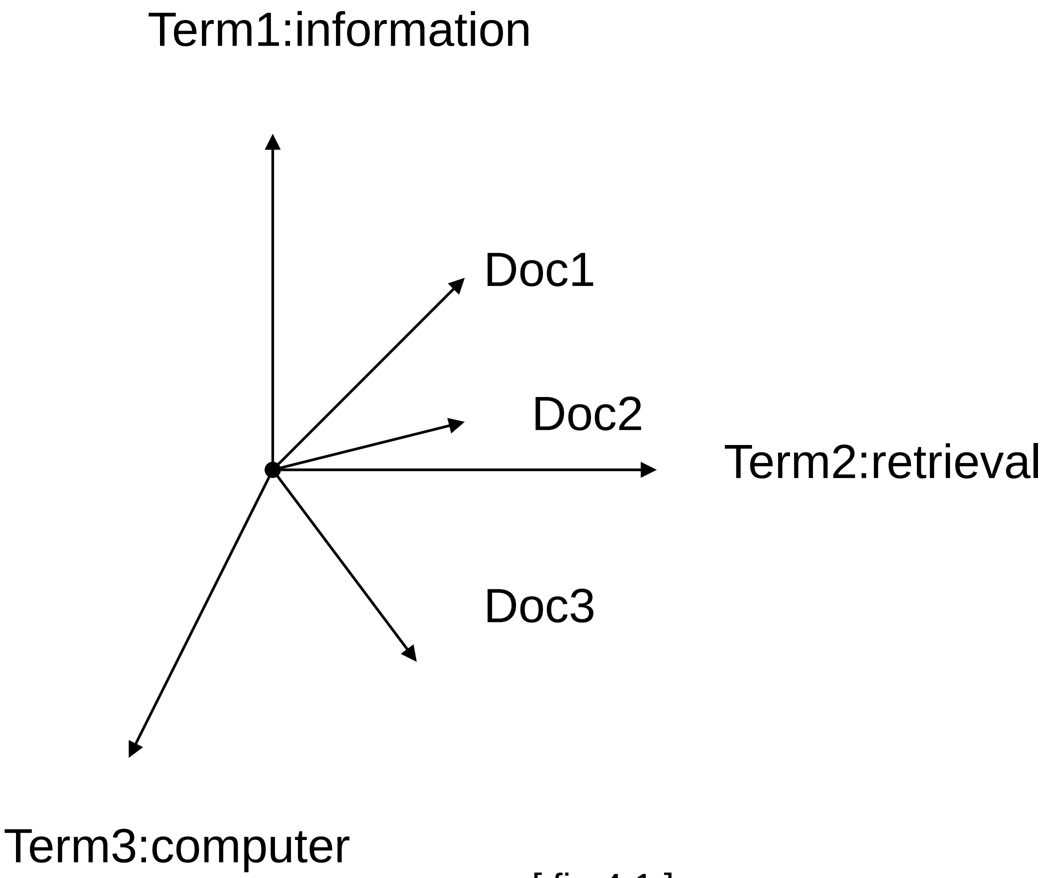
Out[58]:
```
array([0])
```

[fig 3.4]

4. Cosine Similarity :

- Vectors and matrices provide a natural way to represent the occurrence of words in a document/query
- In the text analysis, n is usually the size of the vocabulary, so each dimension corresponds to a unique word
- Doc 1= “information retrieval”
- Doc 2 = “computer information retrieval”
- Doc 3 = “computer retrieval”
- **Vocabulary: information, retrieval, computer**
- Doc 1 = <1, 1, 0>
- Doc 2 = <1, 1, 1>
- Doc 3 = <0, 1, 1>

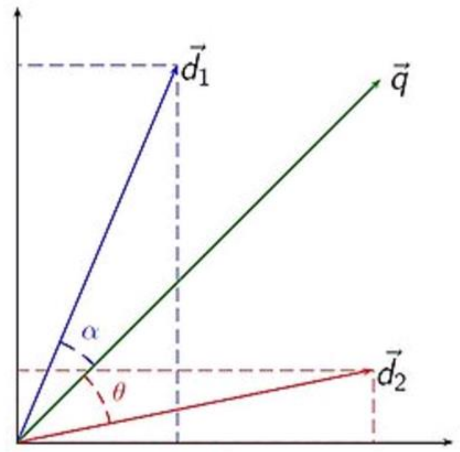
Doc1=<1, 1, 0>
Doc2-<1, 1, 1>
Doc3=<0, 1, 1>



[fig 4.1]

Working with query :

- Used in information retrieval to determine which document (*d_1* or *d_2*) is more similar to a given query *q*
- Documents and queries are represented in the same space.
- Angle (or cosine) is a proxy for similarity between two vectors



[calculating cosine of angle between two vector]

Formula :

- Cos(d1,q) = vector_multiplication(d1,q) / (|d1|*|q|)
- Let’s → d1 = <a1, a2, a3> and q =<b1, b2, b3>
- Cosine(d1,q) = ((a1b1+a1b2+…...+a3b3)/(sqrt(a1a1+a2a2+a3a3)sqrt(b1b1+b2b2+b3b3))
- Similarly we can calculate for Cosine(d2,q)
- Note : Cosine(a,b) = Cosine(b,a) , Cosine(theta) = Cosine(-theta)
- Dis1 = cosine(d1,q)
- Dis2 = cosine(d2,q)
- Compare Dis1 and Dis2, smaller will be nearer to the query vector.

Output Snippet :

```
In [3]: c = 0
        for i in range(len(rvector)):
                c+= l1[i]*l2[i]
        cosine = (d.Decimal(c) / d.Decimal((sum(l1)*sum(l2))**0.5))*100;
```

```
In [5]: with open("model_pickle","rb") as f:
            g1 = pickle.load(f)
        if(g1.predict([sen1]) == g1.predict([sen2]) and cosine < 20):
            cosine = cosine + 25
        elif(g1.predict([sen1]) == g1.predict([sen2]) and cosine < 50):
            cosine = cosine + 20;
        elif(g1.predict([sen1]) == g1.predict([sen2]) and cosine < 70):
            cosine = cosine + 10;
        elif(g1.predict([sen1]) == g1.predict([sen2]) and cosine < 90):
            cosine = cosine + 5;
        print("similarity between ",sen1," and ",sen2," is ", cosine)

        similarity between  a car was baught by me  and  i have baught a car  is  100
```

[fig 4.2]

5. Final Output of Combined Module :

In [22]:
```
def fun(sen1,sen2):
    print("topic decided is for ",sen1," is ",g1.predict([sen1]))
    print("topic decided is for ",sen2," is ",g1.predict([sen2]))
    print("similarity between ",sen1," and ",sen2," is ")
    print((nlp(sen1)).similarity(nlp(sen2)))
    seq1=tokenizer.texts_to_sequences(sen1)
    padded1=pad_sequences(seq1,maxlen=max_len,padding=padding_type, truncating=trunc_type)
    x = model2.predict(padded1)
    print(sen1," from model2  sarcastic percentage is ",model2.predict(padded1)[0][0])
    seq2=tokenizer.texts_to_sequences(sen2)
    padded2=pad_sequences(seq2,maxlen=max_len,padding=padding_type, truncating=trunc_type)
    xx = model2.predict(padded2)
    print(sen2," from model2  sarcastic percentage is ",(model2.predict(padded2)[0][0]))
    dd = pd.DataFrame([[sen1]],columns=['data'])
    dd2 = pd.DataFrame([[sen2]],columns=['data'])
    vec = CountVectorizer(decode_error="replace", vocabulary=vec1)
    for i in model3.predict(vec.transform(dd['data'])):
        if(i == 1 and x[0][0] > 0.4):
            print(sen1," has a negative sentiment")
        else:
            print(sen1," has a positive sentiment")
    for i in model3.predict(vec.transform(dd2['data'])):
        if(i == 1 and xx[0][0] > 0.4):
            print(sen2," has a negative sentiment")
        else:
            print(sen2," has a positive sentiment")
```

In [23]:
```
fun(sen1=input('first sentences : '),sen2=input('second sentences : '))
```

```
first sentences : I have never lost a game, I just ran out of time
 second sentences : we play but we loose
topic decided is for  I have never lost a game, I just ran out of time  is  ['sports']
topic decided is for  we play but we loose  is  ['sports']
similarity between  I have never lost a game, I just ran out of time  and  we play but we loose  is
0.8664395884272713
I have never lost a game, I just ran out of time  from model2  sarcastic percentage is  0.5763317
we play but we loose  from model2  sarcastic percentage is  0.5763317
I have never lost a game, I just ran out of time  has a positive sentiment
we play but we loose  has a negative sentiment
```

[fig 5]

Conclusion :

- Since we have the data from the user from any social networking site eg:twitter,facebook etc, we will be able to analyze the similarity between two comments whether the newly comments are in the context for the older one ,eg : we can consider for tweets and retweets
- we can analyze that the the semantic behavior of two comments are in the same context or not.
- We can also analyze that the older and newly comments are in the same context or not,
- sometimes there is chance that some adds will be there for any specific person which is not meaningful for the older comments.
- We can calculate the cosine similarity between the comments for various page that how much they are similar to one another

Reference :

[1] Arbel, N. (2018, Dec 21). *How LSTM networks solve the problem of vanishing gradients*. Retrieved from medium: https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

[2] Borcan, M. (2020, Jun 8). *TF-IDF Explained And Python Sklearn Implementation*. Retrieved from towardsdatascience: https://towardsdatascience.com/tf-idf-explained-and-python-sklearn-implementation-b020c5e83275

[3] Narkhede, S. (2018, May 17). *Understanding Logistic Regression*. Retrieved from towardsdatascience: https://towardsdatascience.com/understanding-logistic-regression-9b02c2aec102

[4] Pant, A. (2019, Jan 22). *Introduction to Logistic Regression*. Retrieved from towardsdatascience: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

[5] Saha, S. (2018, Dec 15). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Retrieved from towardsdatascience: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[6] Salgado, R. (2020, Apr 16). *Topic Modeling Articles with NMF*. Retrieved from medium: https://medium.com/@robert.salgado

[7] sawant, m. (2019, Jul 28). *Text Sentiments Classification with CNN and LSTM*. Retrieved from medium: https://medium.com/@mrunal68/text-sentiments-classification-with-cnn-and-lstm-f92652bc29fd

[8] *sklearn.decomposition.NMF.* (2007). Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html

[9] *sklearn.feature_extraction.text.TfidfVectorizer.* (2007). Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

THANK YOU