भारतीय प्रौद्योगिकी संस्थान
काशी हिन्दू विश्वविद्यालय

IIT INDIAN INSTITUTE OF TECHNOLOGY
BANARAS HINDU UNIVERSITY

EXPLORATORY PROJECT REPORT

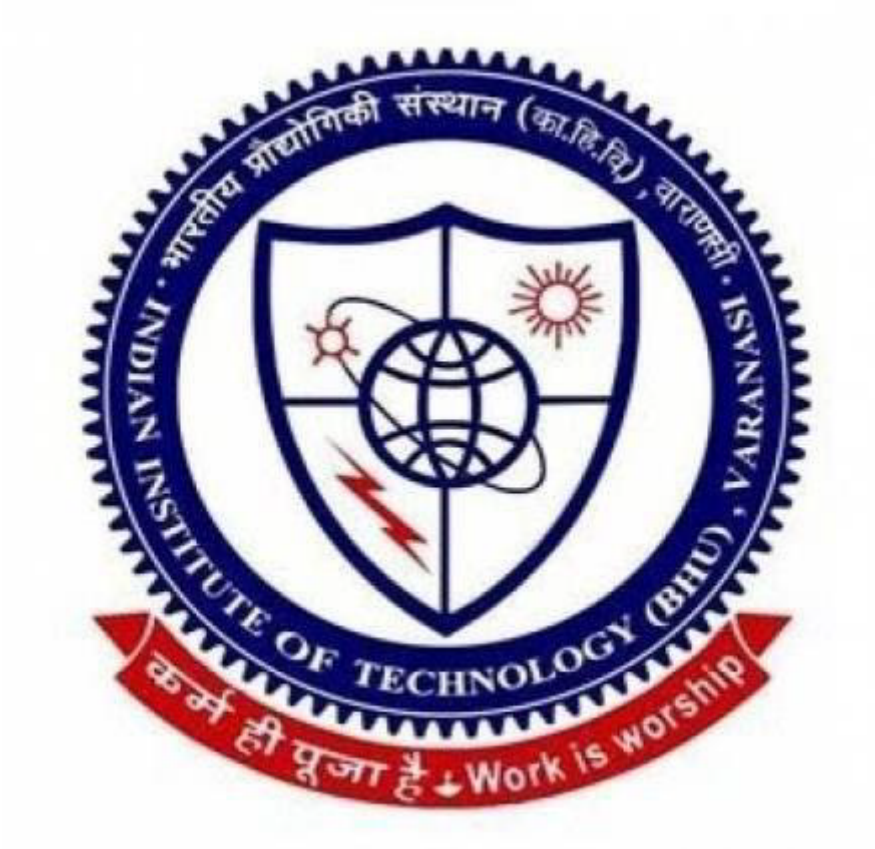
TOPIC

Sentiment Analysis Using Social Media Posts

Project made under guidance of- Dr. Shivam Verma

TEAM MEMBERS-

- **Bisesh Agarwal (20095030)**
- **Mayank Singh (20095065)**
- **Prashasti Tripathi (20095083)**
- **Sanidhya Taparia (20095100)**



ACKNOWLEDGEMENT

We would like to thank our highly respected and esteemed teacher Dr. Shivam Verma, who gave us this opportunity to work on this project. His useful suggestions for this whole work are deeply acknowledged. We got to learn a lot from this project about NLP and ML.

Finally, we would like to extend our heartfelt thanks to all the people who helped and motivated us to finish this project. Without their help this project would not have been successful.

Students Name :

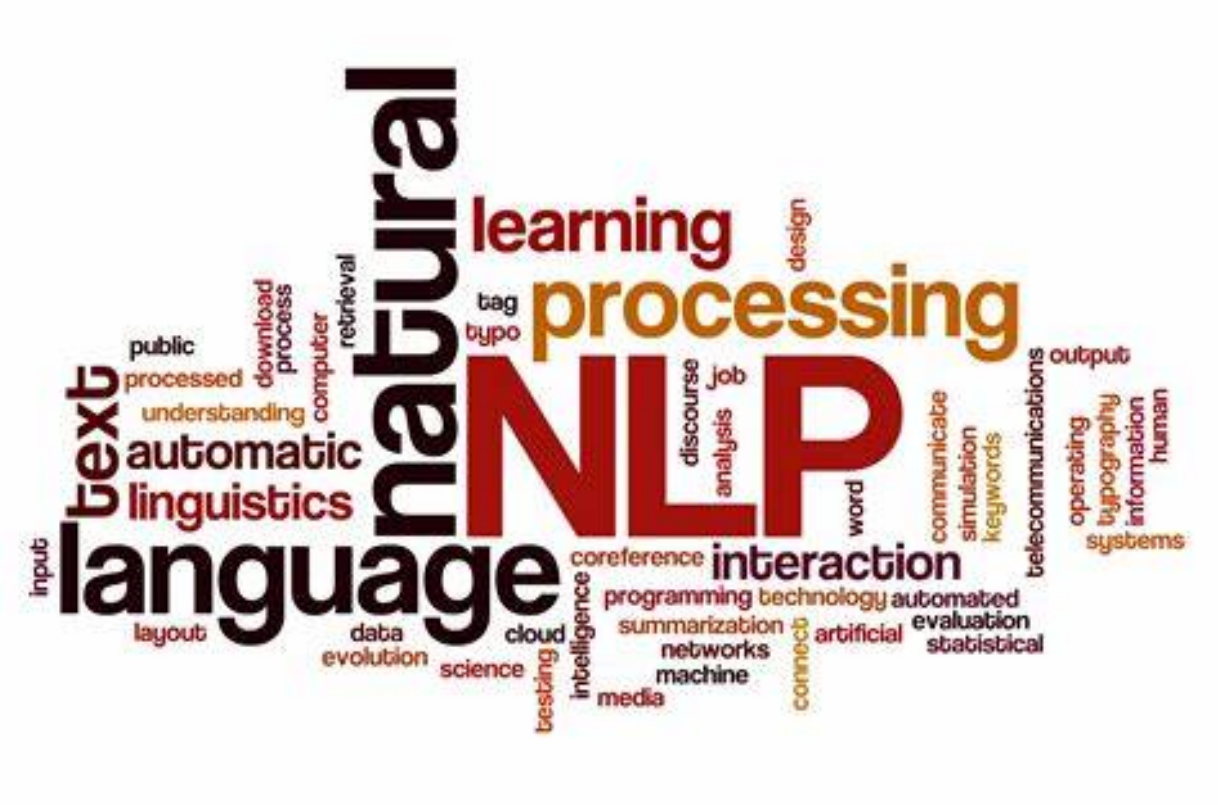
☐ Bisesh Agarwal (20095030)
☐ Mayank Singh (20095065)
☐ Prashasti Tripathi (20095083)
☐ Sanidhya Taparia (20095100)

Date : 2-05-2022

ABSTRACT

Sentiment Analysis is the process of determining the emotional tone behind a series of words, used to gain an understanding of **the attitudes, opinions and emotions expressed within an online mention.**
It is extremely useful in **social media monitoring** as it allows us to gain an overview of the **wider public opinion behind certain topics** like in businesses to learn more about customers and competitors. Sentiment analysis has become an integral part of marketing and can also help government agencies to **analyze mental health problems**.
Sentiment analysis is contextual mining of words which indicates the social sentiment of a brand and also helps the **business to determine whether the product which they are manufacturing is going to make a demand in the market or not**.
Sentiment Analysis uses automation to evaluate texts and detect consumer sentiment implied in them. Rather than reading reams of text, this process employs machine learning and algorithms and categorises the tone of texts as positive, negative, or neutral.
It requires use of NLP (Natural Language Processing), which will be explained further in the report.
So, we decided to take up this area of research which already has a lot of challenges to overcome as our project idea to analyse the working of different statistical machine learning models and neural networks in real life problems and the current automated industry processes with the help of AI (Artificial Intelligence).

NATURAL LANGUAGE PROCESSING



Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can. NLP combines computational linguistics—rule-based modelling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment. NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarise large volumes of text rapidly—even in real time. There's a good chance you've interacted with NLP in the form of voice-operated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences. But NLP also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes.

CHALLENGES FACED

Human language is filled with ambiguities that make it incredibly difficult to write software that accurately determines the intended meaning of text or voice data.

- The main problems that exist in the current techniques are: inability to perform well in different domains, inadequate accuracy and performance in sentiment analysis based on insufficient labelled data, incapability to deal with complex sentences that require more than sentiment words and simple analysing.
- For example-Polarity, words such as “love” and “hate” are high on positive (+1) and negative (-1) scores in polarity. These are easy to understand., but there are in-between conjugations of words such as “not so bad” that can mean “average” and hence lie in mid-polarity (-75). Sometimes phrases like these get left out, which dilutes the sentiment score.
- Tone can be difficult to interpret verbally, and even more difficult to figure out in the written word. Things get even more complicated when one tries to analyse a massive volume of data that can contain both subjective and objective responses. Brands can face difficulties in finding subjective sentiments and properly analysing them for their intended tone.
- The problem with social media content that is text-based, like Twitter, is that they are inundated with emojis. NLP tasks are trained to be language specific. While they can extract text from even images, emojis are a language in itself. Most emotion analysis solutions treat emojis like special characters that are removed from the data during the process of text mining. But doing so means that companies will not receive holistic insights from the data.
- Idioms like actually Machine learning programs don’t necessarily understand a figure of speech. For example, an idiom like “not my cup of tea” will boggle the algorithm because it understands things in the literal sense. Hence, when an idiom is used in a comment or a review, the sentence can be misconstrued by the algorithm or even ignored. To overcome this problem a sentiment analysis platform needs to be trained in understanding idioms. When it comes to multiple languages, this problem becomes manifold .Likewise there exists many more problems in this domain.

DATASETS USED AND PRE-PROCESSING

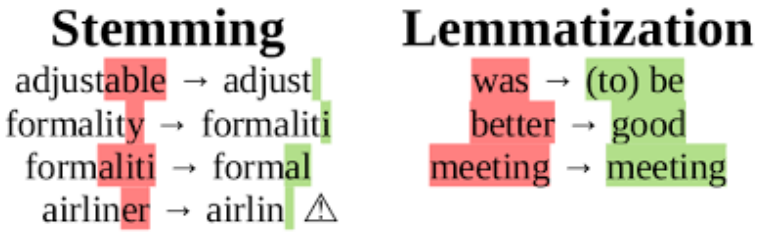
We have used the twitter sentiment analysis dataset (**Sentiment140 dataset**) which is available on kaggle.
First we have **Cleaned** and organised the data into something more readable and manageable and then **Preprocessed** the data such that the observations (reviews) are more concise, and more obviously positive or negative.
Text files are usually not the easiest to clean due to the flexibility of language. However, there're a few standard ways to clean text data.

1. Filter out symbols (eg. question marks, full stops, etc)
2. Remove stop words (ie. the non-polarising words like "wasn't", "shan't", etc)
3. Lemmatising or Stemming

Symbols: Symbols may play a role in segmenting paragraphs or helping with grammar; however, they do not hold a high value when it comes to keeping the essence of the sentence. Hence, removing them would be helpful to the model.

Stop words: Stop words seem to those common words that mainly help with connecting other kinds of words together to form sentences, or something to that extent.The book "Introduction to Information Retrieval" that describes stop words as "... extremely common words which would appear to be of little value ...".

Lemmatization and Stemming: These are methods used to convert words into their common base form; or grouping together words that have essentially the same meaning. For example, if we were to lemmatise or stem these words
['connect', 'connected', 'connection', 'connects']
We'd get the word '**connect**' for all of those instances.

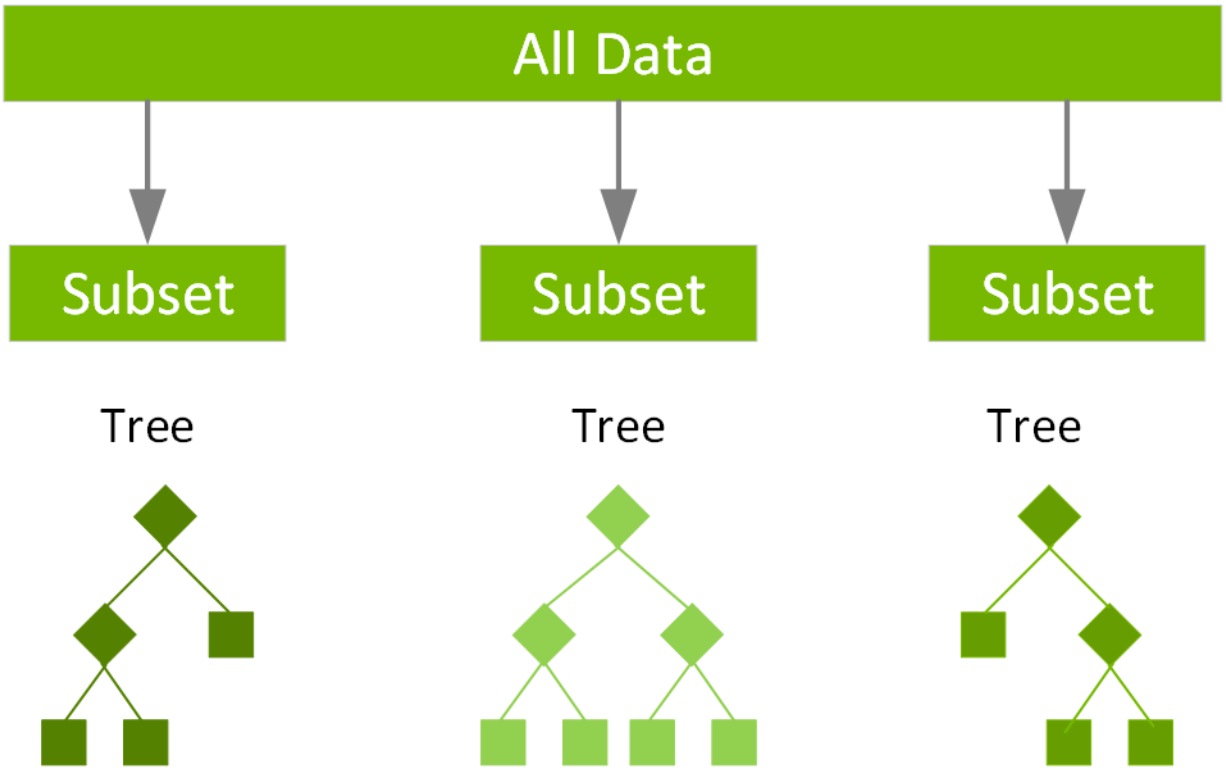


MODELS AND TECHNIQUES USED

1.XGBoost

Basic Idea Of XGBoost Algorithm:

XGBoost, which stands for Extreme Gradient Boosting, is an implementation of Gradient Boosted decision trees. ***XGBoost models majorly dominate in many Kaggle Competitions.*** In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.



We will be using XGBoost as a classifier to complete the task of distinguishing Positive from Negative tweets. This algorithm takes a text string as an input and loads an XGBoost model trained on the Dataset taken to predict its sentiment.

Preprocessing Part:

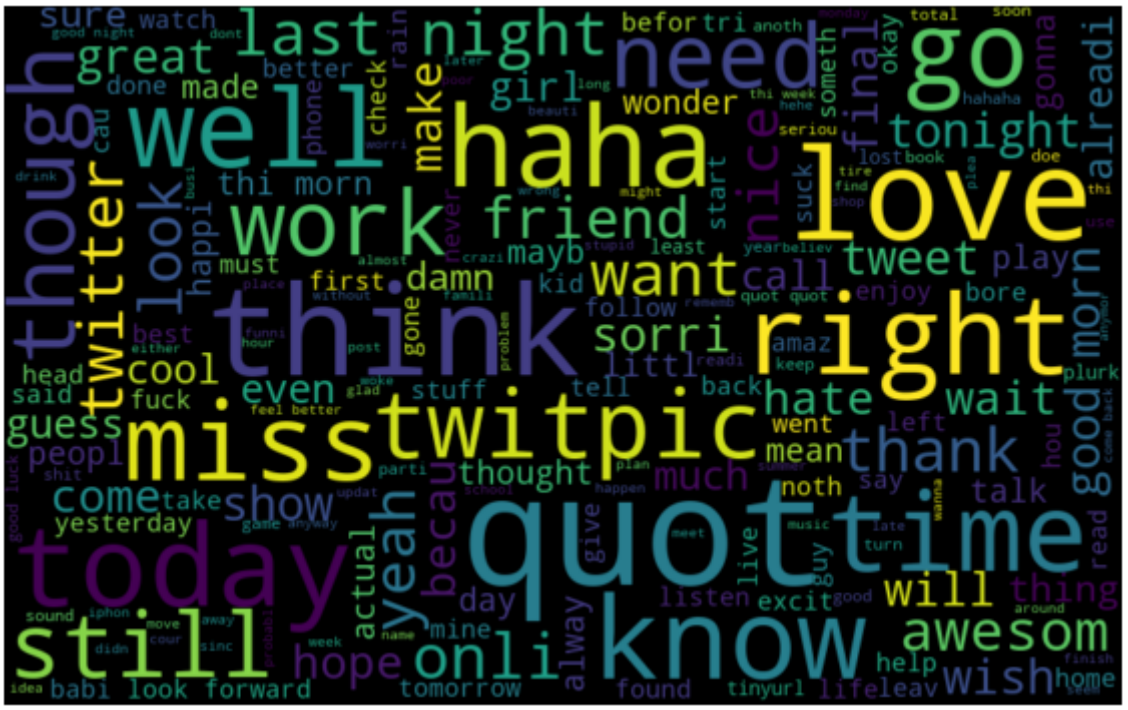
- Before fitting an XGBoost classifier on this data, first the texts are preprocessed to remove the English stop words, short words (length<3) and the punctuations(like @,#,?,!,.).

- Before we vectorize the text we need to tokenize it and then normalise it. Further we called a Function to extract the content of Hashtags, because it also describes the quality of a tweet. Whereas a Tag(@) does not describe the quality of a tweet so we will ignore it.

- After that, to be able to feed the text data as numeric values to our model which is called **Word Embedding**.They are first converted into a matrix of token counts and we are going to use **Bag of Words (bow)** using a CountVectorizer. Then this count matrix is converted to a normalised tf-idf (term-frequency times inverse document-frequency) representation. Using this transformer, we are scaling down the impact of tokens that occur very frequently, because they convey less information to us since they are very common and they don't help us make the differentiation. On the contrary, we are scaling up the impact of the tokens that occur in a small fraction of the training data because they are more informative to us.

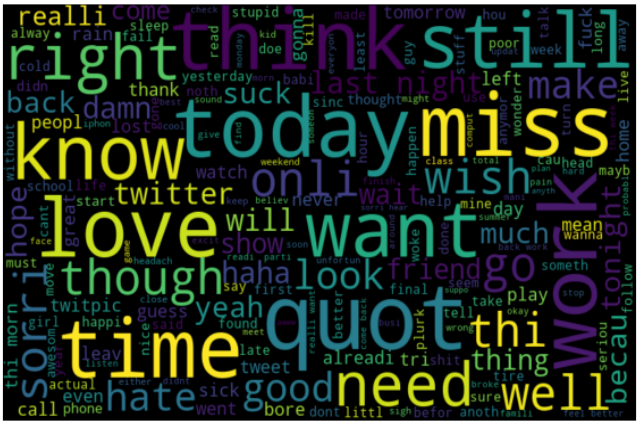
WordCloud:

All Common Words

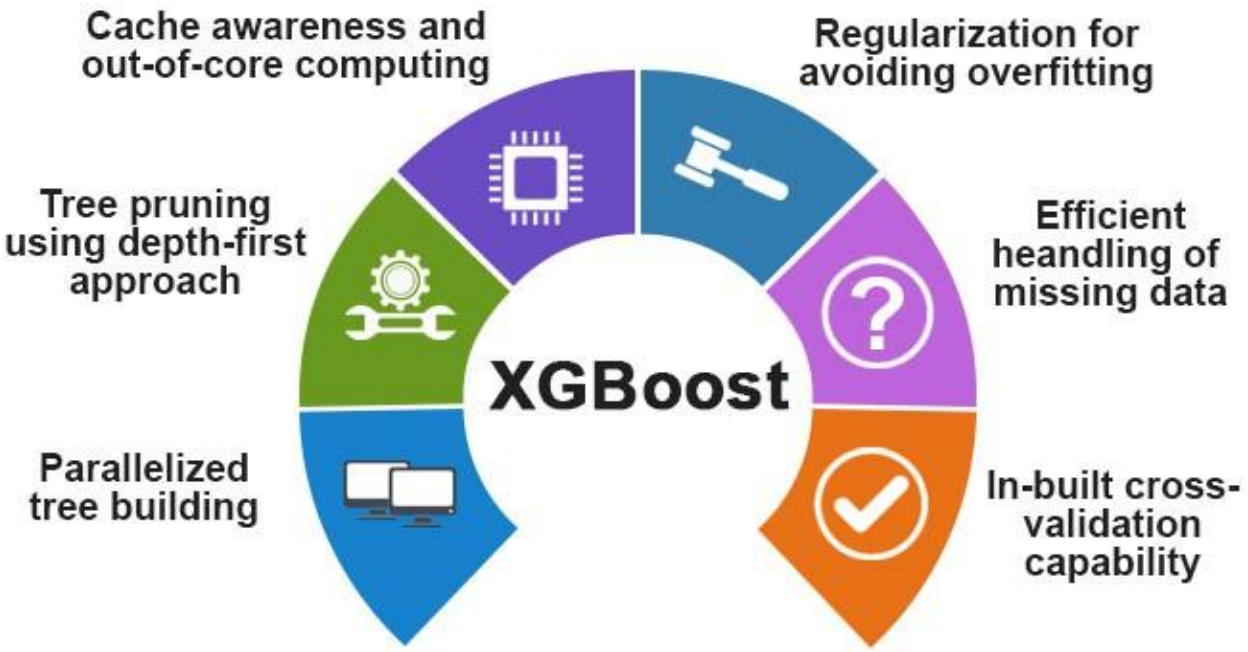


Positive Common Words

Negative Common Words



Why XGBoost?



Implementing a Machine Learning Model:

Now that the data is set up, I could focus on fitting this data into a machine learning model. There are a number of algorithms that we can use to train our model but in this part we are going to use XGBoost. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost stands for eXtreme Gradient Boosting. XGBoost performs better than most predictive models. It is for these reasons that we are going to be using it to classify our tweets. We chose this model using multiple simple decision tree models, thereafter collating and boosting them all to achieve a single collated result. Using an XGBoost model requires the dataset to be in an even more specific state. Of course, it is not that simple as randomly filling in the parameters and then using the model. But the general idea in this Exploratory Project is to create a machine learning model class, tweak its parameters to fit the dataset, and thereafter train it, and then use it to predict your test set.

The code implementation is shown below.

```
# Bag of words features

# Extracting train and test BoW features

train_bow = bow[:1600000,:]
test_bow = bow[1600000:,:]
# print(train_bow.shape[0])
train_bow.shape[0]==data['target'].shape[0]
# # splitting data into training and validation set
xtrain_bow, xvalid_bow, ytrain, yvalid = train_test_split(train_bow, data['target'], random_state=42,test_size=0.3)
```

```
[23] # XGB Classifier on training data(15m)
from sklearn.metrics import f1_score, accuracy_score
from xgboost import XGBClassifier
xgb_model = XGBClassifier(max_depth=6, n_estimators=1000).fit(xtrain_bow, ytrain)
prediction = xgb_model.predict(xvalid_bow)
f1_score(yvalid, prediction)
```

```
0.7557303266518189
```

```
# # XBG Classifier on test data
# xgb_model = XGBClassifier(max_depth=6, n_estimators=1000).fit(xtrain_bow, ytrain)
prediction = xgb_model.predict(xtrain_bow)
f1_score(ytrain, prediction)
```

```
0.7598386994939006
```

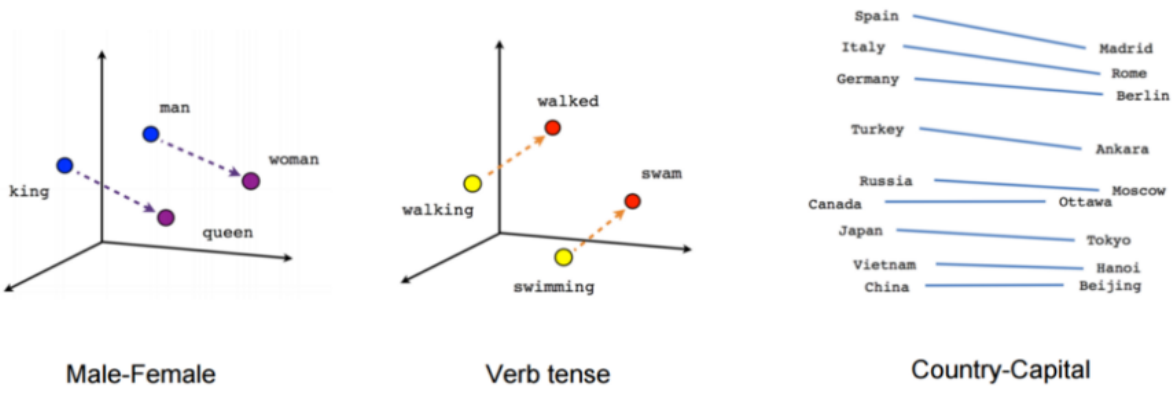
2.Embedding + Stacked-LSTM

OVERVIEW

Before diving into the model, let us understand the meaning of embedding and stacked LSTM.

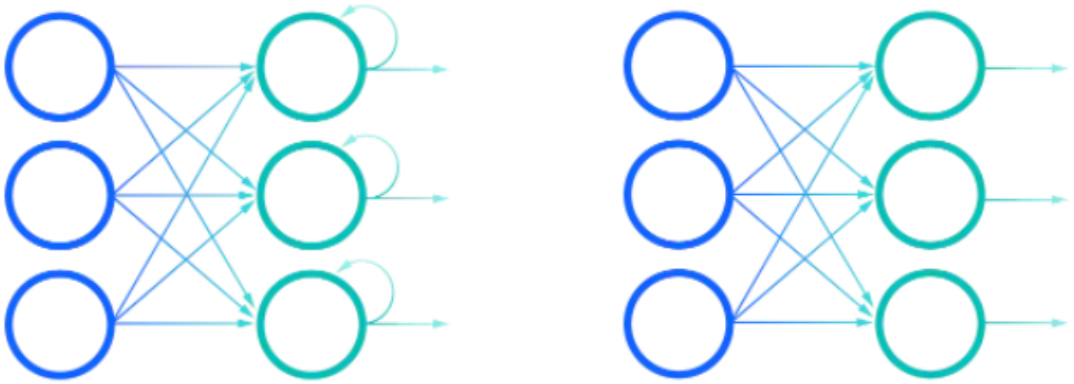
- **Word Embedding:** A word embedding is a learned representation for text where words that have the same meaning have a similar representation.Here, each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning.

  For example, words like **“mom”** and **“dad”** should be closer together than the words **“mom”** and **“ketchup”** or **“dad”** and **“butter”**.



Male-Female | Verb tense | Country-Capital

- **LSTM:** LSTM (Long Short Term Memory Network) is an advanced type of RNN (Recurrent neural network). RNN is a type of artificial neural network which uses sequential data or time series data.While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence.

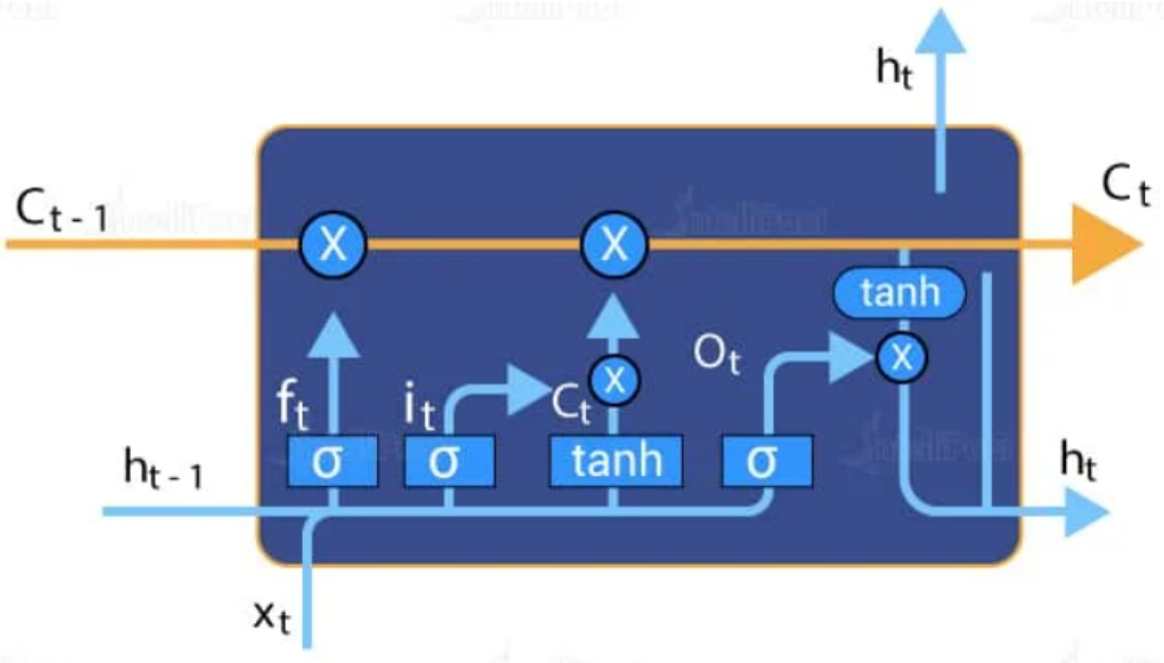


Comparison of Recurrent Neural Networks (on the left) and Feedforward Neural Networks (on the right)

Now let us understand how LSTM use this special “memory” feature while training the model….

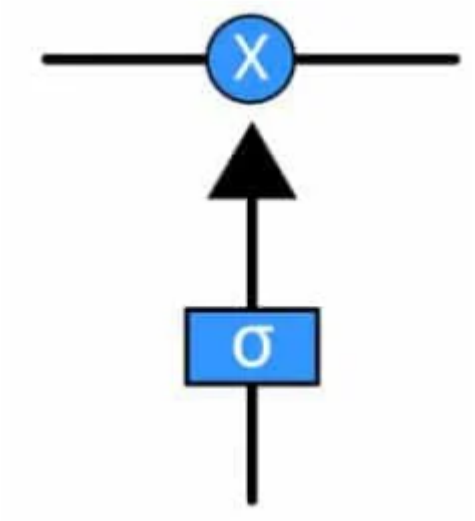
The Logic Behind LSTM

The central role of an LSTM model is held by a memory cell known as a ‘cell state’ that maintains its state over time. The cell state is the horizontal line that runs through the top of the below diagram. It can be visualised as a conveyor belt through which information just flows, unchanged.



Information can be added to or removed from the cell state in LSTM and is regulated

by gates. These gates optionally let the information flow in and out of the cell. It contains a pointwise multiplication operation and a sigmoid neural net layer that assist the mechanism.



The sigmoid layer gives out numbers between zero and one, where zero means ‘nothing should be let through,’ and one means ‘everything should be let through.’

STEPS INVOLVED:

- **Tokenization:** It is a particular kind of document segmentation. It breaks up text into smaller chunks or segments called tokens. A tokenizer breaks unstructured data, natural language text, into chunks of information that can be counted as discrete elements. After this operation these counts of token occurrences in a particular document can be used as a vector representing a given document.
- **Word embeddings using GloVe:** Embedding matrix is used in embedding layer in the model to embed a token into its vector representation, that contains information regarding that token or word.

  Here, we have taken the embedding vocabulary from the tokenizer and the corresponding vectors from the embedding model, which in this case is the GloVe model.
  GloVe stands for Global Vectors for Word Representation and it is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics

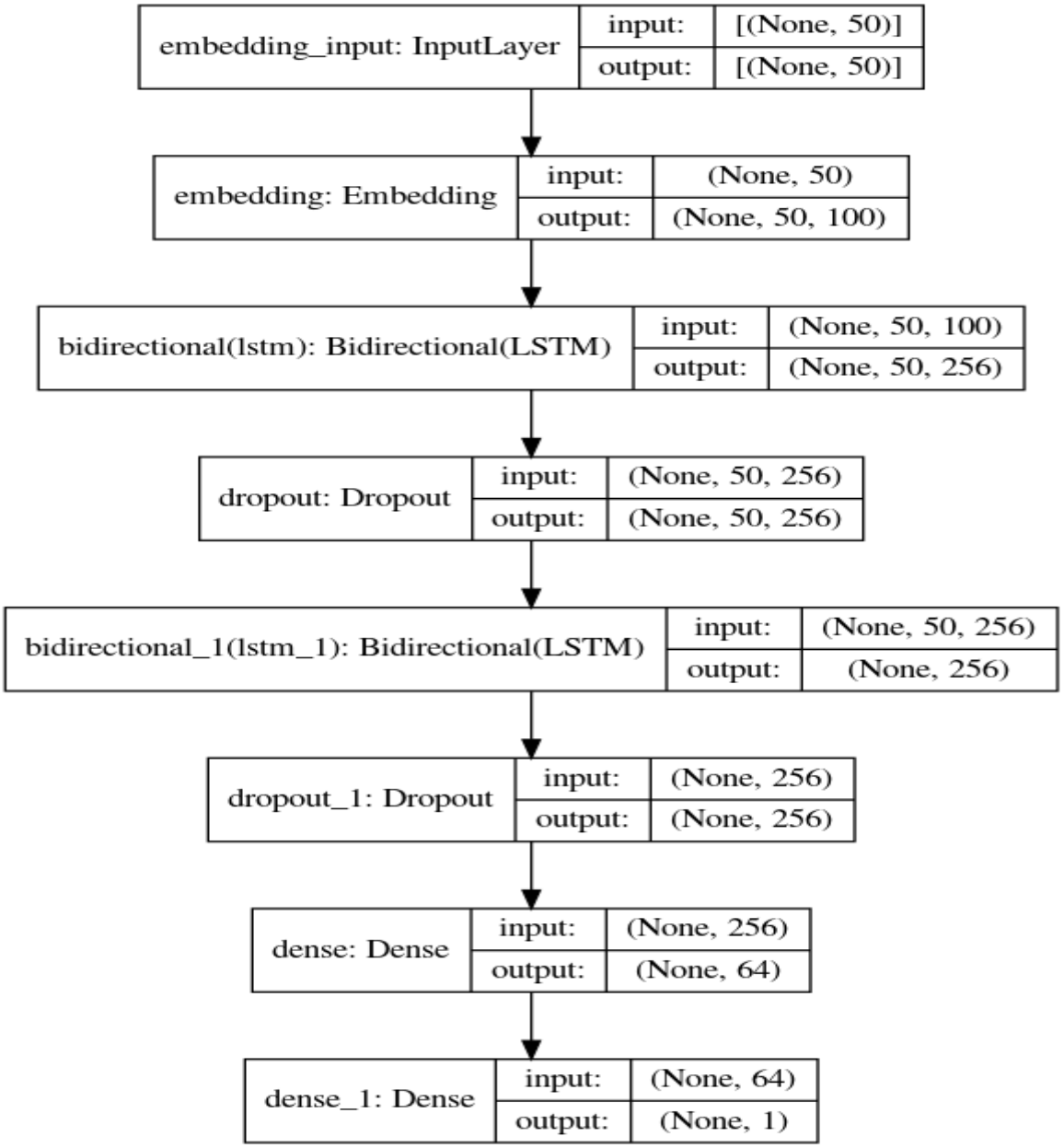
from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Below is the code part used for embedding using GloVe:

```
embeddings_dictionary = dict()
embedding_dim = 100
glove_file = open('/kaggle/input/glove6b100dtxt/glove.6B.100d.txt'

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = np.asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions

glove_file.close()

embeddings_matrix = np.zeros((vocab_size, embedding_dim))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embeddings_matrix[index] = embedding_vector
```

- **Model Training:** The model consisted of two bidirectional LSTM layers, two Dropout layers, one relu dense layer, and one sigmoid dense layer.

  The overall model structure can be visualised as:


embedding_input: InputLayer
input: [(None, 50)]
output: [(None, 50)]
embedding: Embedding
input: (None, 50)
output: (None, 50, 100)
bidirectional(lstm): Bidirectional(LSTM)
input: (None, 50, 100)
output: (None, 50, 256)
dropout: Dropout
input: (None, 50, 256)
output: (None, 50, 256)
bidirectional_1(lstm_1): Bidirectional(LSTM)
input: (None, 50, 256)
output: (None, 256)
dropout_1: Dropout
input: (None, 256)
output: (None, 256)
dense: Dense
input: (None, 256)
output: (None, 64)
dense_1: Dense
input: (None, 64)
output: (None, 1)

CONCLUSION

The model gave an overall validation accuracy of 77% with a macro-average of 77% and also weighted average of 77% which can be seen in the below output printed using **classification_report** function of **skLearn.**

	precision	recall	f1-score	support
0	0.78	0.75	0.76	79800
1	0.76	0.79	0.77	80200
accuracy			0.77	160000
macro avg	0.77	0.77	0.77	160000
weighted avg	0.77	0.77	0.77	160000

3.Fine Tuning Bert for Sentiment Analysis

OVERVIEW

One of the biggest milestones in the evolution of NLP recently is the release of Google's BERT, which is described as the beginning of a new era in NLP. We have created a baseline model, in which we have used a TF-IDF vectorizer and a Naive Bayes classifier and got accuracy of **78.18%**. The transformers library can help us fine-tune the state-of-the-art BERT model and yield an accuracy rate **10%** higher than the baseline model.

NAIVE BIAS CLASSIFIER:

What is a classifier?

A classifier is a machine learning model that is used to discriminate different objects based on certain features.

Principle and uses of Naive Bayes Classifier:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification tasks. The crux of the classifier is based on the Bayes theorem.
Naive Bayes algorithms are mostly used in sentiment analysis, spam filtering, recommendation systems etc. They are fast and easy to implement but their biggest disadvantage is that the requirement of predictors to be independent. In most of the real life cases, the predictors are dependent, this hinders the performance of the classifier.

Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

TF-IDF VECTORIZER:

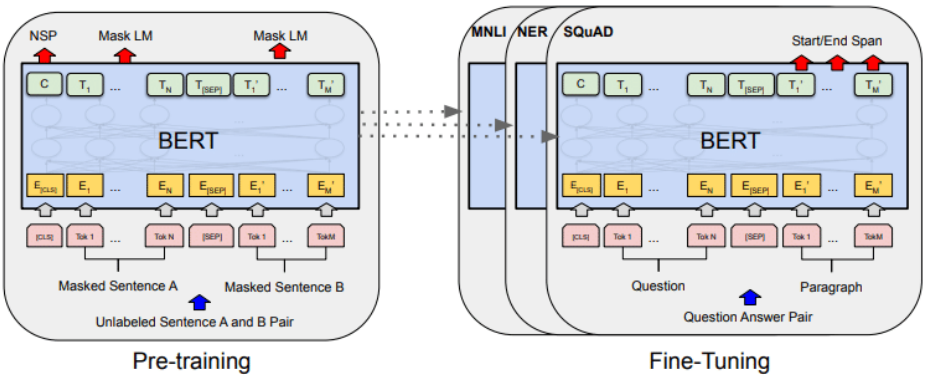
In information retrieval, **TF-IDF**, short for **term frequency–inverse document frequency**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. We will use TF-IDF to vectorize our text data before feeding them to machine learning algorithms.

BERT Model:

BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers; was proposed by Google AI in the paper, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". This is one of the groundbreaking models that has achieved the

state of art in many downstream tasks and probably one of the most exciting developments in NLP in recent years.

BERT makes use of **Transformer**, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an **encoder** that reads the text input and a **decoder** that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism of the Transformer is necessary.



CONCLUSION:

The accuracy could have been increased upto 10% by the BERT model.

FINAL VERDICT:

We used 3 models namely: **XGBoost**, **LSTM** and **Naive Bias** and it can be seen that the highest accuracy achieved till now was by naive bias approach....and could have been improved by 10 percent with the use of Transformers.

CODE LINKS:

- Dataset
- Preprocessing
- XGboost
- Embedding+LSTM
- Naive Bias

REFERENCES

- XGBoost1
- XGBoost2
- LSTM1
- LSTM2
- Naiv Bias1
- Naive Bias2