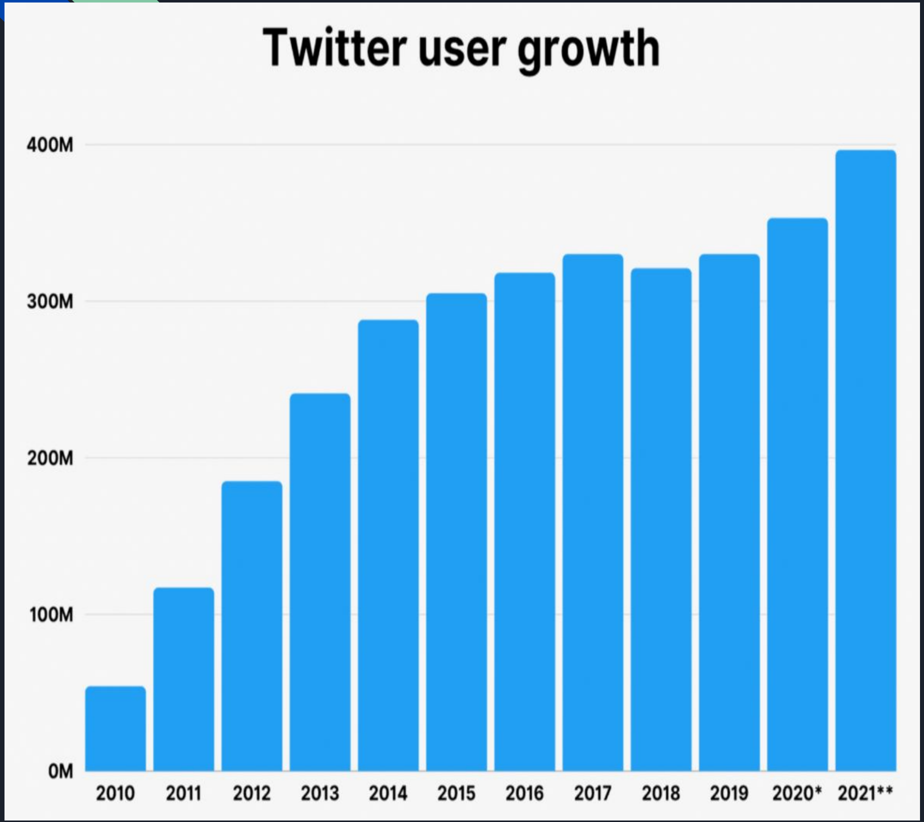
Twitter Sentiment Analysis

Ashit Neema

Vedang Gharat

Kartik Saini

The Purpose



- First Assignment was based on extracting tweets and analyzing it
- Twitter has 396.5 million users worldwide
- With 6k tweets posted per second, the platform is generating huge amount of data
- Opportunity to study user opinions, POV, and sentiments on various topics and events

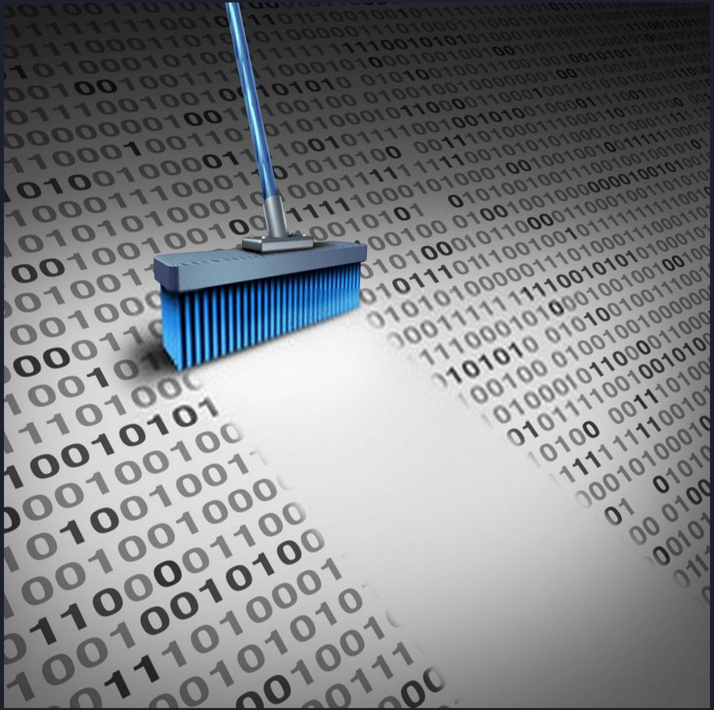
Introduction

- Sentiment analysis is one of the applications of Natural Language Processing
- People tend to share their sentiment in their writings - We have to identify it and then classify it
- We are classifying a piece of text as positive, negative or neutral
- We are aiming to develop a model based on Keras Sequential Model along with Word2vec vectorizer
- Due to the presence of too much noise in the data, our main focus is to preprocess and clean the data
- The performance of this model is then calculated using F1 score

About dataset

- This is the sentiment140 dataset
- It contains 1,600,000 tweets extracted using the twitter api
- Contains the following 6 fields:
    1. Sentiment : Polarity of the tweet
    2. Id_number : ID
    3. Date : Date of the tweet
    4. Query : The query
    5. User_name : Username
    6. Tweet_text : Text of the tweet

Data Preprocessing / Data Cleaning



- Many words and characters not really required
- Remove the twitter handles
- Get rid of the punctuations, numbers and even special characters
- Normalize the text data. For ex- reducing terms like loves, loving, and lovable to their base word, i.e., ‘love’
- Convert the text to lowercase, remove punctuations, remove special characters.
- Filter out useless data / useless words also known as Stop words. NLTK has a list of stop words
- Stemming - remove ing, ly, etc
- Tokenize the text - Easy for feature extraction

```
Sample Text before cleaning :

@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer.  You shoulda got David Carr of Third Day to do it. ;D
After cleaning:
awww bummer shoulda got david carr third day



Another example of Text before cleaning :

is upset that he can't update his Facebook by texting it... and might cry as a result  School today also. Blah!
After cleaning:
upset update facebook texting might cry result school today also blah
```

Feature Extraction

- To analyse a preprocessed data, it needs to be converted into features
- Text features can be constructed – Bag of Words, TF-IDF, and Word Embeddings

Bag of words / TF

- Count the frequency of every unique words in the document
- Dimensionality problem - The more the number of different words, the bigger the matrix

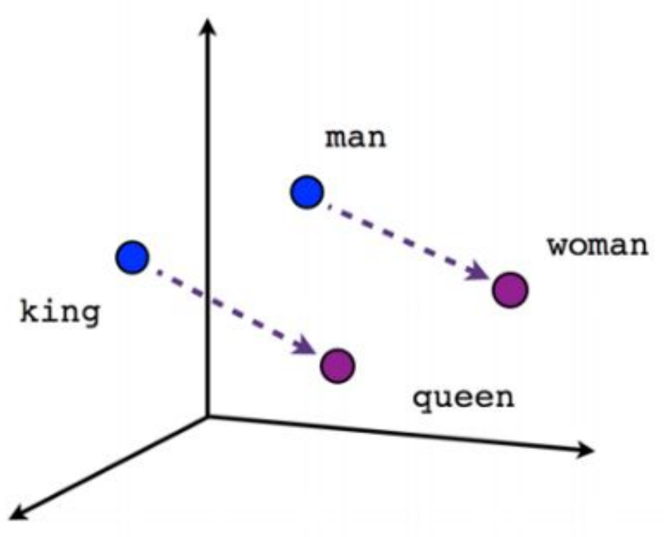
Advance Bag of words / TF-IDF

- Term-frequency - The term frequency of a word in a document
- Inverse document frequency: How common or rare a word is in the entire document set

Word embeddings / Word 2 vec

- Words are mapped to vectors of real numbers.
- The modern way of representing words as vectors
- Objective - redefine the high dimensional word features into low dimensional feature vectors

Word2vec embeddings



Male-Female

- It's a combination of two techniques – CBOW and Skip-gram model
- CBOW predict the probability of a word given a context
  A context may be a single adjacent word or a group of surrounding words
- The Skip-gram model works in the reverse manner, it tries to predict the context for a given word
- Three layers in W2V - an input layer, a hidden layer, and an output layer
- Skip-gram advantage over CBOW - It can capture two semantics for a single word
- Word2Vec model will convert all the unique words to vectors
- The advantages of using word embeddings over BOW or TF-IDF are

  Dimensionality reduction - significant reduction in the no. of features required to build a model

  It capture meanings of the words, semantic relationships and the different types of contexts they are used in

Example from our training

Similarity of the word “hate” after training our word2vec model

```
[('hates', 0.5370832681655884),
 ('sucks', 0.4964931309223175),
 ('suck', 0.4752613306045532),
 ('hating', 0.47210004925727844),
 ('stupid', 0.4718897044658661),
 ('dislike', 0.45377856492996216),
 ('h8', 0.43292367458343506),
 ('annoying', 0.3989093601703644),
 ('ugh', 0.39854592084884644),
 ('horrible', 0.3861214816570282)]
```

Keras Tokenizer

fit_on_texts

- First, this method creates the vocabulary index based on word frequency
- Every word gets a unique integer value. 0 is reserved for padding
- Lower integer means more frequent words

texts_to_sequences

- Transforms each text to a sequence of integers
- It takes each word in the text and replaces it with its corresponding integer value from the word_index dictionary

Keras Sequential Model

```
#Creating the architecture of our model
embedding_layer = Embedding(number_of_words, 300, weights=[weight_matrix], input_length=300, trainable=False)
model = Sequential()
model.add(embedding_layer)
model.add(Dropout(0.5))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a gen
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 300, 300)          87118500

 dropout (Dropout)           (None, 300, 300)          0

 lstm (LSTM)                 (None, 100)               160400

 dense (Dense)               (None, 1)                 101
```

Result and Accuracy

	precision	recall	f1-score	support
NEGATIVE	0.79	0.79	0.79	160340
POSITIVE	0.79	0.80	0.79	159660
accuracy			0.79	320000
macro avg	0.79	0.79	0.79	320000
weighted avg	0.79	0.79	0.79	320000

```
] accuracy_score(y_labels, y_predicted)

0.790475
```

F1 score and Accuracy

```
[ ] predict("I love cricket")

[0.9334277]
{'Given_Text': 'I love cricket',
 'Predicted_Sentiment': 'POSITIVE',
 'Text_Score': 0.9334276914596558}

[ ] predict("I hate politics")

[0.05876258]
{'Given_Text': 'I hate politics',
 'Predicted_Sentiment': 'NEGATIVE',
 'Text_Score': 0.058762576431035995}

[ ] predict("I am speechless")

[0.5194313]
{'Given_Text': 'I am speechless',
 'Predicted_Sentiment': 'NEUTRAL',
 'Text_Score': 0.5194312930107117}
```

Prediction

Future Work

- Implementing Transformers like BERT. They have a superior performance in many natural language processing tasks

- Multilingual comment classification is still a challenge

- We can boost the efficiency by implementing different models - Logistic Regression, Naive Bayes (NB), SVM, Xgboost, Stochastic Gradient Descent (SGD)

QUESTIONS ? SUGGESTIONS ?