What is Sentiment Analysis?

	Reviews	
Fragnance-1 (Lavender)	1. smell amazing 2. must buy super amazing.	Positive
	Review	
Fragnance-2 (Rose)	1. A decent purchase 2. Quite Okayish.	Neutral
	Review	
Fragnance-3 (Lily)	1. Smell okay 2. decent purchase	Negative

Sentiment Analysis is a use-case of NLP and comes under the category of text-classifica

Sentiment Analysis involves classifying a text

eg. Dataset,

SNo.	Student feedback	Sentiment label
1.	timing are very odd such course shouldn't be offered	negative
2.	very hard working instructor helps a net.	positive
3.	Give more programming assign.	neutral

How does Sentiment Analysis Work?

1. Text - Preprocessing
2. Tokenization
3. Feature Extraction
4. Sentiment Classification

Types of Sentiment Analysis

- Document - level
- Sentence - level
  - → objective (fact)
  - → subjective (views of other)
- Aspect - level

Feature Extraction

- Vector Representation
- Positive & Negative Count

How does sentiment Analysis work?

(1) Text preprocessing → This steps involves cleaning the text data by removing noise, such as punctuation, special characters, and irrelevant information like URL's or HTML tag

It may also involve tasks like Lowercasing, removing stopwords (commonly used words like "the", "is", "and" etc) and handling word contractions.

(2) Tokenization → In this step, the preproce text is split into smaller units called tokens, which are typically words or phrases Tokenization makes it easier to analyze and process the text data.

There are different methods of tokenization such as word tokenization which splits the text into individual words, and sentence tokenizati which splits the text into sentences.

(3) Feature Extraction → Once, the text has bee tokenized, features are extracted from the tokens to represent the text data in a format that can be used by ML algorithms

(4) Sentiment Analysis → finally, ML models or statistical techniques are applied to the extracted features to classify the sentiment of the text. Common approaches includes SVM, logistic, deep learning, RNNs, CNNs.

NLTK → Natural Language Toolkit.

Preprocessing : remove stopwords and punctuations.

stopwords	punctuations
and	.
is	,
are	:
at	;
has	!
for	?
a	

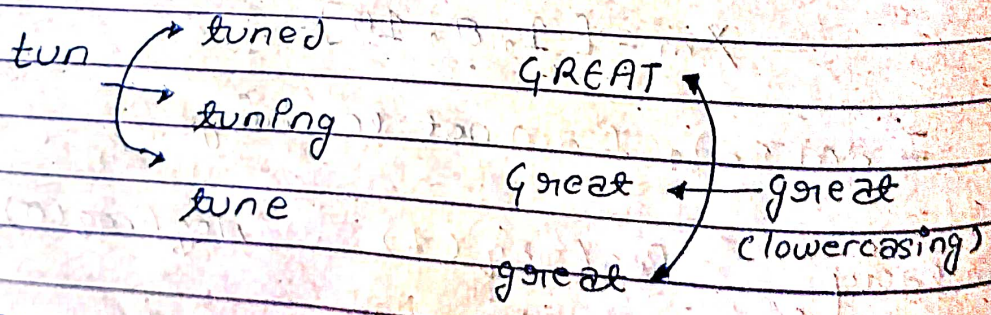
@Yhouri and @Andrew Ng are tuning

Handles and URL's →

~~@Yhouri and @Andrew Ng are tuning~~

Stemming & Lowercasing →

tuning great AI model

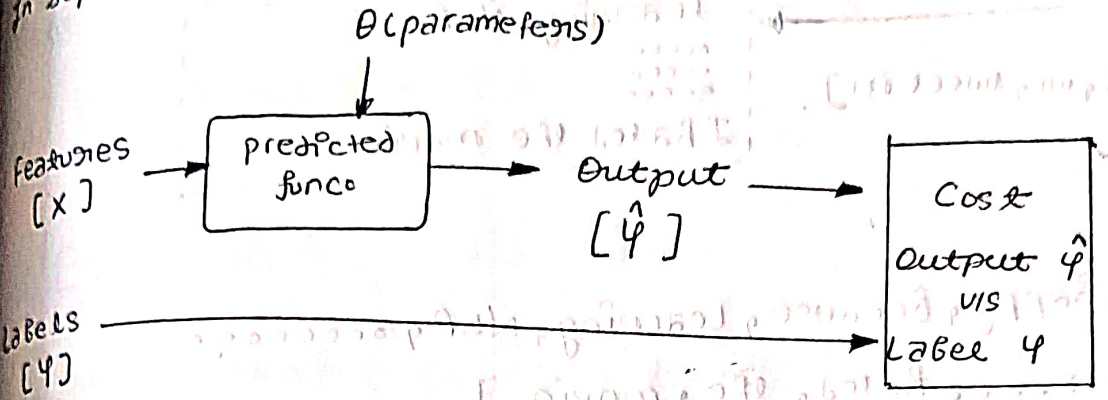


preprocessed tweet ↓

[tun, ai, great, model].

Logistic Regression for Sentiment Analysis of tweets:

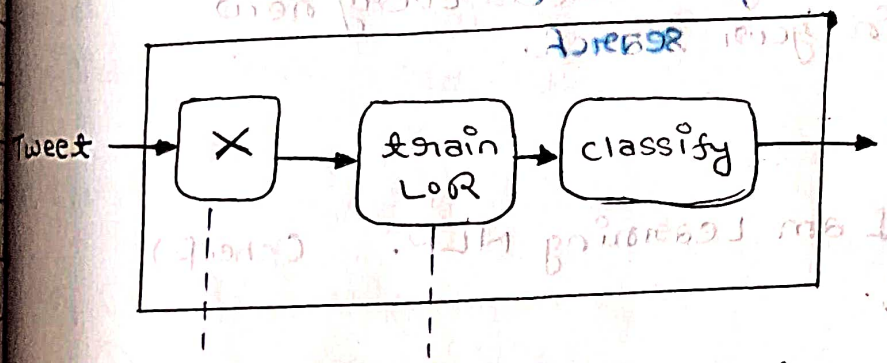
In supervised ML,



Whether the tweet have a positive or negative sentiment?

Tweet: I am happy because I am learning NLP → positive sentiment: 1

→ Negative sentiment: 0



Extract useful features — train your Logistic reg. classifier — while minimizing the costs

1. Vocabulary & feature extraction:

How to represent a text as a vector →

First build a vocabulary, that will allow you to encode any text or any tweet as an array of numbers.

Vocabulary

Tweets → I am happy because I am learning NLP. ..... I hated the movie.

[tweet 1, tweet 2, ..., tweet m]

V = [I, am, happy, because, learning, NLP, ... hated, the, movie]

- Vocabulary (V) is the list of unique words from your list of tweets.
- To get that list you'll have to go through all the words from ~~the list~~ all your tweets every new word that appears in your search.

Feature Extraction →

I am happy because I am learning NLP. (check)

↓

V = [I, am, happy, because, learning, NLP, ---, hated, the, movie].

↓

- To do so, you will have to check every word from your vocabulary appears in the tweet. you will assign a value to that feature. If it doesn't appear you assign a value

↓

[I, am, happy, because, learning, NLP, ..., hated, the, movie]

[1, 1, 1, 1, 1, 1, ..., 0, 0, 0].

6 count of 1's, many 0's.

Problems with "Sparse" Representations —

A lot of zeros → that's a sparse representation.

Eg I am happy because I am learning NLP

↓

[1, 1, 1, 1, 1, 1, ..., 0, 0, 0].

This representation would have a number of features equal to the size of your entire vocabulary

This would have a lot of features equal to 0 for every tweet

With the sparse representation, a logistic regression model would have to learn (n+1) parameters, where n = size of your vocabulary.

$n = |V| \rightarrow [\theta_0, \theta_1, \theta_2, \ldots, \theta_n]$

Hence, ① Large training time

② Large prediction time.

Positive and Negative frequencies —

Corpus

I am happy Because I am learning NLP.
I am happy
I am sad, I am not learning NLP.
I am sad.

Vocabulary

I
am
happy
Because
learning
NLP
sad
not

Positive tweets

I am happy because I am learning NLP
I am happy

Negative tweets

I am sad, I am not learning NLP.
I am sad.

Vocabulary	PosFreq(1)	NegFreq(0)
I	3	3
am	3	3
happy	2	0
because	1	0
learning	1	1
NLP	1	1
sad	0	2
not	0	1

- previously we encoded a ~~vector~~ tweet as a vector of dim |V|. Now, we'll learn to encode a tweet or specially represented as a vector of dimension 3.

freqs : dictionary mapping from (word, class) to frequency

$$X_m = [1, \sum_w freqs(w, 1), \sum_w freqs(w, 0)]$$

- X_m : arbitrary tweet m or features of tweet m
- 1 : Bias
- $\sum_w freqs(w, 1)$: sum of pos freqs
- $\sum_w freqs(w, 0)$: sum of neg freqs

I am sad, I am not learning NLP.

↓

I am ~~happy~~ ~~because~~ learning NLP sad not

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

3 + 3 + ✗ + ✗ + 1 + 1 + 0 + 0 ≡ 8

I am ~~happy~~ ~~because~~ learning NLP sad not

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

3 + 3 + ✗ + ✗ + 1 + 1 + 2 + 1 ≡ 11

$X_m \equiv [1, 8, 11]$.

tting it all together →

am Happy Because i am Learning NLP @deeplearning

↓ [preprocessing]

[happy, learn, nlp].

↓ [feature extraction]

bias ← [1, 4, 2]

↑ posfreq ↑ Negfreq

⇓

I am Happy Because i am
learning NLP
@deeplearning

am sad not learning NLP

...

am sad :(

→ [happy, learn, nlp]
[sad, not, learn, nlp]
.....
[sad]

→ [[1, 40, 20],
[1, 20, 50],
....
[1, 5, 35]]

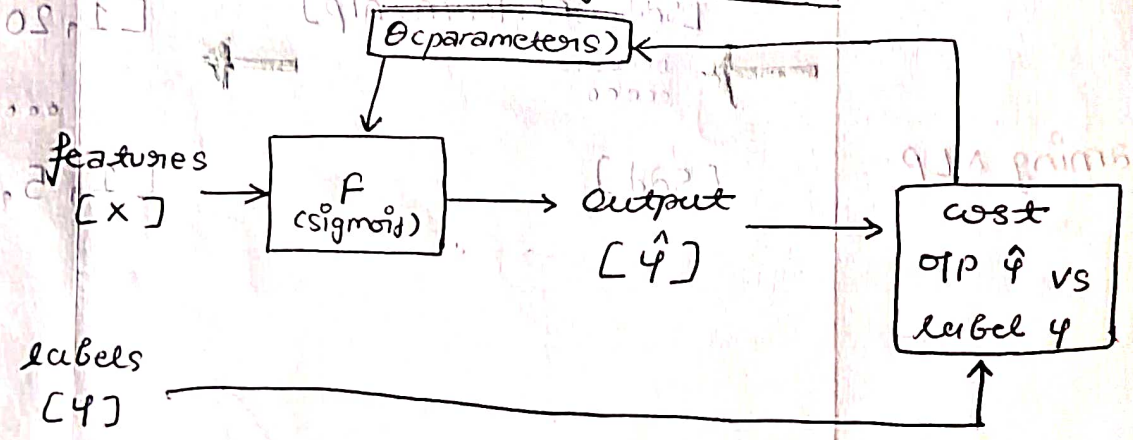
$$\begin{bmatrix} 1 & X_1^{(1)} & X_2^{(1)} \\ 1 & X_1^{(2)} & X_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & X_1^{(m)} & X_2^{(m)} \end{bmatrix} \longleftrightarrow$$

[[1, 40, 20],
[1, 20, 50],
....
[1, 5, 35]].

General Implementation :

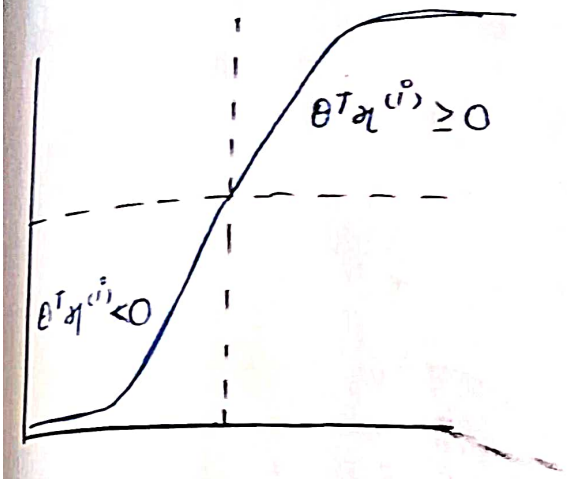
```
freqs = build_freqs(tweet, labels)
# Build frequencies dictionary

X = np.zeros((m, 3)) # Initialize matrix X

for i in range(m): # for every tweet

    p_tweet = process_tweet(tweets[i])
                # process tweet

    X[i, :] = extract_features(p_tweet, freqs)
                # Extract features
```

Overview of Logistic regression →



- to make prediction based on your data, we use a func (sig with some parameters (θ) to map your features to o/p label
- To get an optimum mapping from your features to labels, you minimize the cost function, which works by compari how closely your output ŷ is to the true labels y fro your data.
- After, which the parameters are updated and you repeat process until your cost is minimized for logistic regres this function

$$h(x^{(i)}, \theta) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$$



@YMoussi and
@AndrewYNg are tuning a GREAT AI model

↓

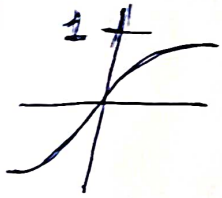
[fun, ai, great, model]

↓

$$x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix}$$

Sum of all pos & neg freqs of all the words in your processed tweets

↓



$$\theta = \begin{bmatrix} 0.00003 \\ 0.00150 \\ -0.00120 \end{bmatrix}$$