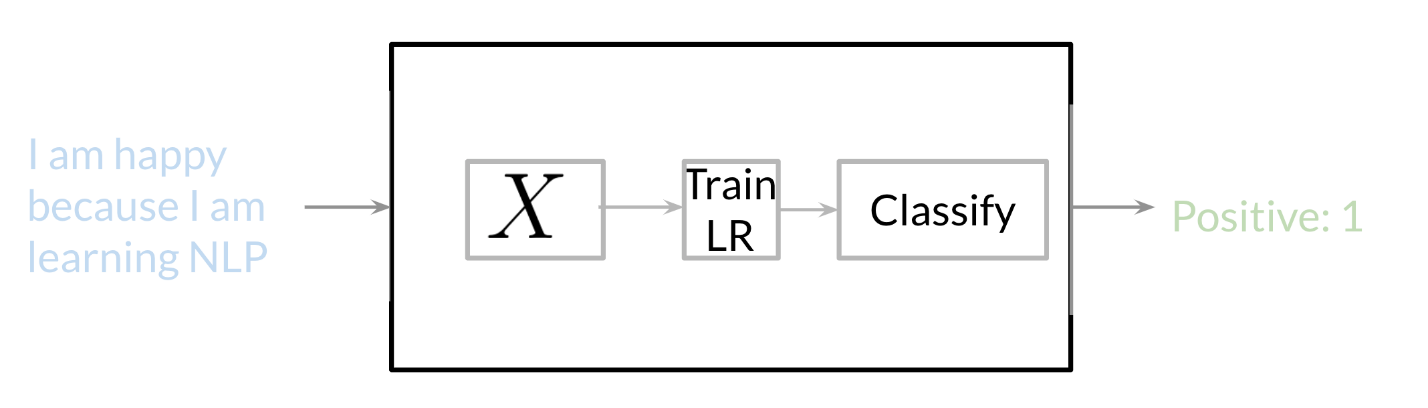
**NLP**

**Sentiment analysis and supervised ML** (logistics regression classifier)

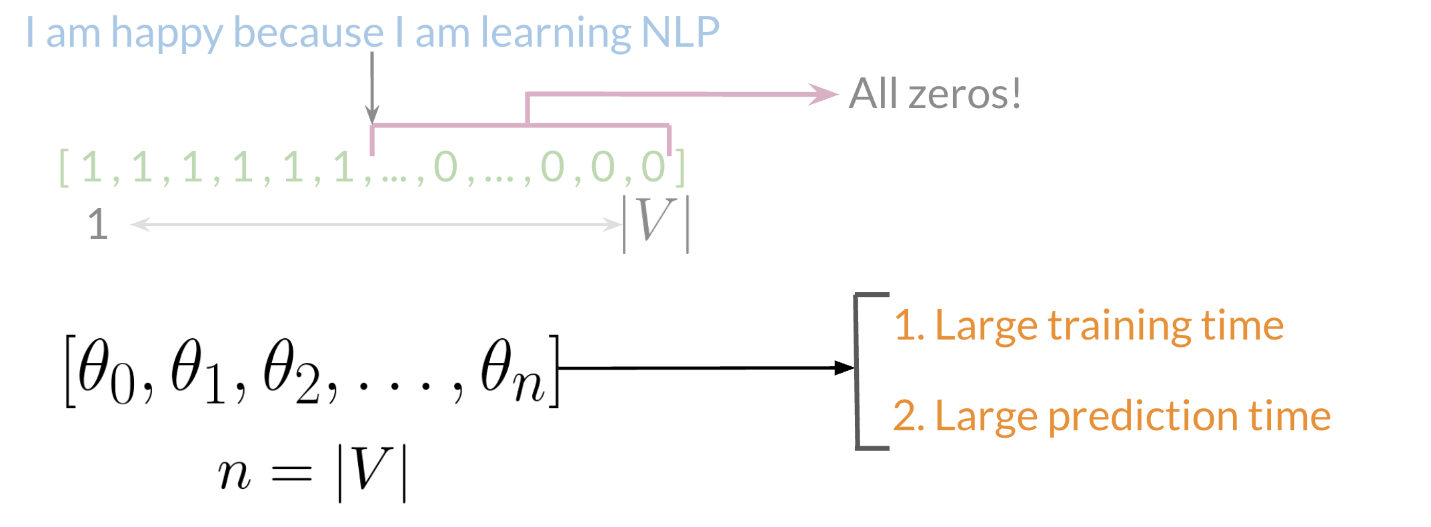
Step 1: extract the features

Step 2: train the model

Step 3: classify the tweet based on the trained model



**Vocabulary & Feature Extraction**



V = vector

# Negative and Positive Frequencies

# Mapping\

# https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/vhHO7A7dTvuRzuwO3U779Q_5364f83a7bd54782a09279efe06e96f2_Screen-Shot-2020-09-01-at-7.57.30-AM.png?expiry=1631923200000&hmac=AZ9Jce0DomohnlEWIGJ_mRAIRsW6jySDr8a_rNvl_UE

# Feature Extraction with Frequencies

# We used dimension of v (vector) now we will use dimensions of 3

# Dimensions of 3 is faster for logistics regression, instead of learning V features we are learning 3 features

# https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/N_PzQkNvSNqz80JDb-ja5A_a44a87942c5e476593cd8e1582cf5b06_Screen-Shot-2020-09-01-at-8.04.10-AM.png?expiry=1631923200000&hmac=ZJ6_CtcV-Pv_3O24o9EI4i5jzggHcnIVc3Mu-ubPy6k

# Preprocessing

# Stemming and Stop-word

When preprocessing, you have to perform the following:

1. Eliminate handles and URLs (doesn’t have any value)
2. Tokenize the string into words.
3. Remove **stop words** like "and, is, a, on, etc."
4. **Stemming-** or convert every word to its stem. Like dancer, dancing, danced, becomes 'danc'. You can use **porter stemmer** to take care of this.
5. Convert all your words to **lower case. Eg:** GREAT ,Great ,great are all same

# Putting it all together

# Over all, you start with a given text, you perform preprocessing, then you do feature extraction to convert text into numerical representation

# Then we make a matrix

# https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/C3BC-AlDRj-wQvgJQyY_8w_6b189b0ef72e456b9cce8c264796f567_Screen-Shot-2020-09-01-at-8.24.17-AM.png?expiry=1631923200000&hmac=k6FmSiDllj0_qSfsUeC_nWjsrplTMRsLD_f6w61j-Tc

# When implementing it with code, it becomes as follows:

# https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/NwmjyOjKQTaJo8joyqE2Ow_330a68fd246c44c3a5409216b096559c_Screen-Shot-2020-09-01-at-8.20.48-AM.png?expiry=1631923200000&hmac=lyjpfx0tJx-MK7COxRCqgK9B4IrFCqY5IojbI1RKpOY

# Logistic Regression Overview

# https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/oL4Ox_JxTBi-DsfycUwYvw_d0582a0dddf7470486f0955c8b025dd6_Screen-Shot-2020-09-01-at-8.30.00-AM.png?expiry=1631923200000&hmac=GJIXNcoKOekaQ1l4V-Mp_p6ESWxM_QDsLZPK-9mfTEY

# Logistic Regression: Training

# You initialize your parameter \theta*θ*, that you can use in your sigmoid, you then compute the gradient that you will use to update \theta*θ*, and then calculate the cost. You keep doing so until good enough.

# https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/YGmjEyR0Sw2poxMkdBsNeQ_74cb9a1075fb4d1eb835b14a8d5b2456_Screen-Shot-2020-09-01-at-8.39.39-AM.png?expiry=1631923200000&hmac=8vfHJvGCjVg7dbNW1phQkwg33-2AMDXoF4spaMzKfkU

# Logistic Regression: Testing

# https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/xq8RYoHvROKvEWKB73TiUg_ac2e78d0c6654f58ab40822d08b68465_Screen-Shot-2020-09-02-at-10.47.33-AM.png?expiry=1631923200000&hmac=_0Z99r3P_kyofNcDB4wnFegMufSA2isQ12jLulXohno

# https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/1NW_9EVkS6yVv_RFZGusFA_90304c432911444eb5f981a4aaa97c47_Screen-Shot-2020-09-02-at-10.53.31-AM.png?expiry=1631923200000&hmac=7eS2rid-j8rNZaUPlr2VyBYvXVYTO5IbSr95RO3Rkh8

# Logistic Regression: Cost Function

# https://d3c33hcgiwev3.cloudfront.net/imageAssetProxy.v1/zL6sM-PyTsO-rDPj8r7DrA_89c49e130c854fd3b754a3bcc234897e_Screen-Shot-2020-09-02-at-11.26.30-AM.png?expiry=1632009600000&hmac=x6SQf6ZbPieCx4cGvB2j6NVOlg8Ky5PQ55Vayg7jeVY

The logistic regression cost function is defined as

J(\theta)=-\frac{1}{m} \sum\_{i=1}^{m}\left[y^{(i)} \log h\left(x^{(i)}, \theta\right)+\left(1-y^{(i)}\right) \log \left(1-h\left(x^{(i)}, \theta\right)\right)\right]*J*(*θ*)=−*m*1​∑*i*=1*m*​[*y*(*i*)log*h*(*x*(*i*),*θ*)+(1−*y*(*i*))log(1−*h*(*x*(*i*),*θ*))]

As you see from the image when theta is 1, you predict something close to 0 the value is infinity and on the other hand when theta is 0, you predict something close to 1 the value is infinity. On the other hand when theta is close to label J(theta) is 0 In either case we are trying to get close to 0 that is trying to minimize J(theta)

Code:

import re  
import string  
from nltk.corpus import stopwords  
from nltk.stem import PorterStemmer  
from nltk.tokenize import TweetTokenizer  
import numpy as np

Pre

#Preprocessing tweets  
def process\_tweet(tweet):  
#Remove old style retweet text "RT"  
tweet2 = re.sub(r'^RT[\s]','', tweet)  
  
#Remove hyperlinks  
tweet2 = re.sub(r'https?:\/\/.\*[\r\n]\*','', tweet2)  
  
#Remove hastags  
#Only removing the hash # sign from the word  
tweet2 = re.sub(r'#','',tweet2)  
  
# instantiate tokenizer class  
tokenizer = TweetTokenizer(preserve\_case=False, strip\_handles=True, reduce\_len=True)  
  
# tokenize tweets  
tweet\_tokens = tokenizer.tokenize(tweet2)   
  
#Import the english stop words list from NLTK  
stopwords\_english = stopwords.words('english')   
  
#Creating a list of words without stopwords  
tweets\_clean = []  
for word in tweet\_tokens:  
if word not in stopwords\_english and word not in string.punctuation:  
tweets\_clean.append(word)  
  
#Instantiate stemming class  
stemmer = PorterStemmer()  
  
#Creating a list of stems of words in tweet  
tweets\_stem = []  
for word in tweets\_clean:  
stem\_word = stemmer.stem(word)  
tweets\_stem.append(stem\_word)  
  
return tweets\_stem