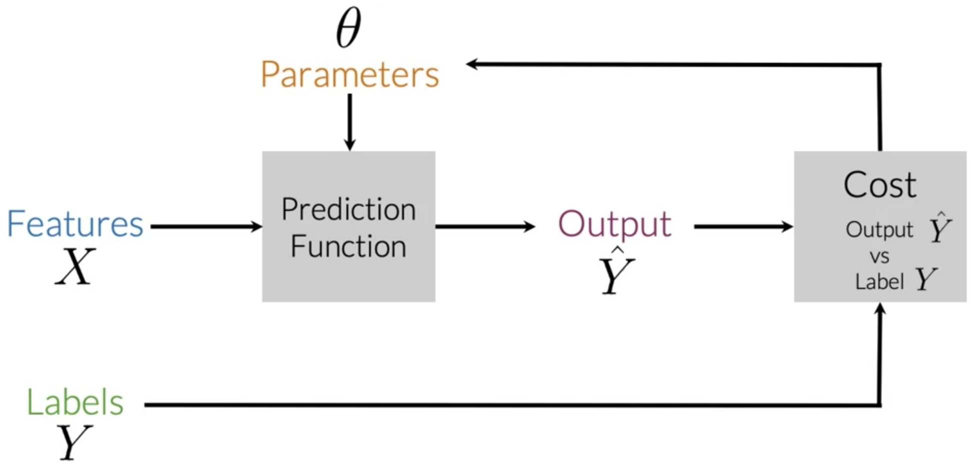
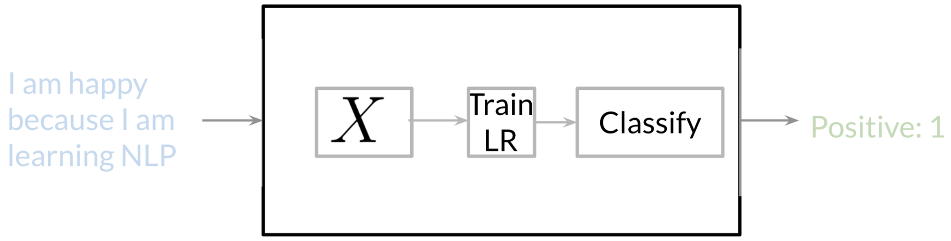
Week 1

1. Supervised ML & Sentiment Analysis

In supervised machine learning, you usually have an input X, which goes into your prediction function to get your $\hat{Y}$. You can then compare your prediction with the true value Y. This gives you your cost which you use to update the parameters θ. The following image, summarizes the process.



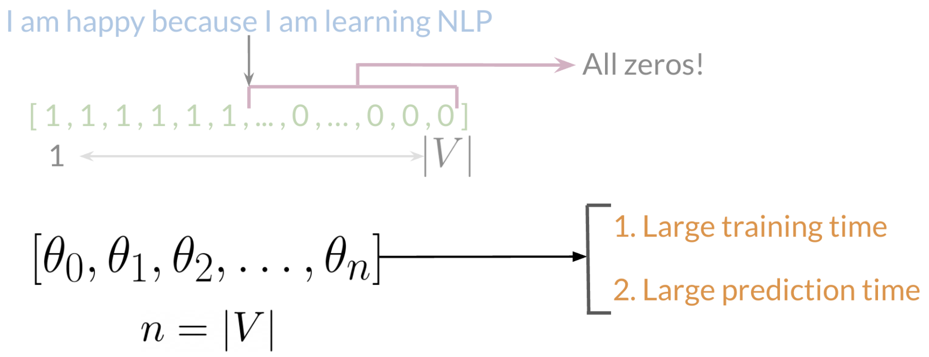
To perform sentiment analysis on a tweet, you first have to represent the text (i.e. "I am happy because I am learning NLP ") as features, you then train your logistic regression classifier, and then you can use it to classify the text.



Note that in this case, you either classify 1, for a positive sentiment, or 0, for a negative sentiment.

2. Vocabulary & Feature Extraction

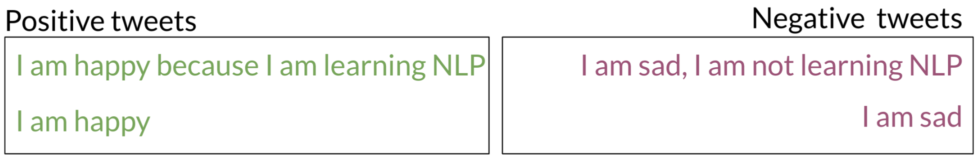
Given a tweet, or some text, you can represent it as a vector of dimension V, where V corresponds to your vocabulary size. If you had the tweet "I am happy because I am learning NLP", then you would put a 1 in the corresponding index for any word in the tweet, and a 0 otherwise.



As you can see, as V gets larger, the vector becomes more sparse. Furthermore, we end up having many more features and end up training θ V parameters. This could result in larger training time, and large prediction time.

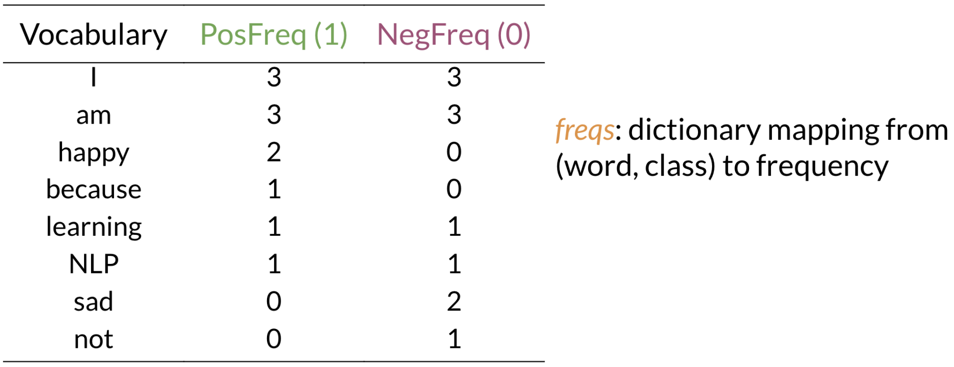
3. Feature Extraction with Frequencies

Given a corpus with positive and negative tweets as follows:



Positive tweets	Negative tweets
I am happy because I am learning NLP	I am sad, I am not learning NLP
I am happy	I am sad

You have to encode each tweet as a vector. Previously, this vector was of dimension V. Now, as you will see in the upcoming videos, you will represent it with a vector of dimension 3. To do so, you have to create a dictionary to map the word, and the class it appeared in (positive or negative) to the number of times that word appeared in its corresponding class.



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

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

In the past two videos, we call this dictionary `freqs`. In the table above, you can see how words like happy and sad tend to take clear sides, while other words like "I, am" tend to be more neutral. Given this dictionary and the tweet, "I am sad, I am not learning NLP", you can create a vector corresponding to the feature as follows:

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

I am sad, I am not learning NLP

$$X_m = [1, \underset{\substack{\big\downarrow \\ \boxed{8}}}{\sum_w freqs(w,1)}, \sum_w freqs(w,0)]$$

To encode the negative feature, you can do the same thing.

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

I am sad, I am not learning NLP

$$X_m = [1, \sum_w freqs(w,1), \underset{\substack{\big\downarrow \\ \boxed{11}}}{\sum_w freqs(w,0)}]$$

Hence you end up getting the following feature vector $[1, 8, 11]$. 11 corresponds to the bias, 88 the positive feature, and 1111 the negative feature.

4. Preprocessing

When preprocessing, you have to perform the following:

1. Eliminate handles and URLs

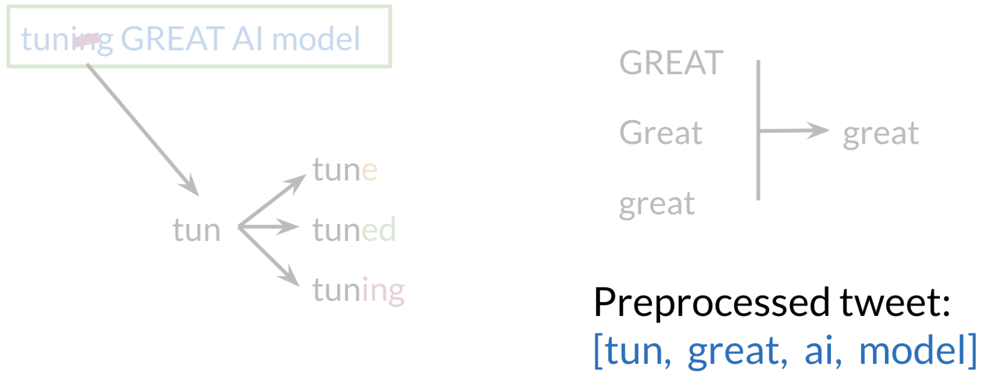
2. Tokenize the string into words.

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.

For example the following tweet "@YMourri and @AndrewYNg are tuning a GREAT AI model at https://deeplearning.ai!!!" after preprocessing becomes

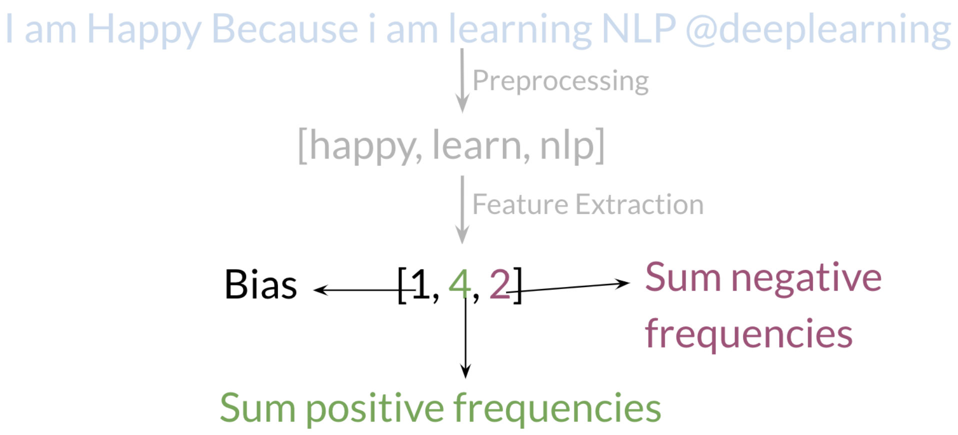


Preprocessed tweet:
[tun, great, ai, model]

$[tun, great, ai, model]$. Hence you can see how we eliminated handles, tokenized it into words, removed stop words, performed stemming, and converted everything to lower case.

5. Putting it all together

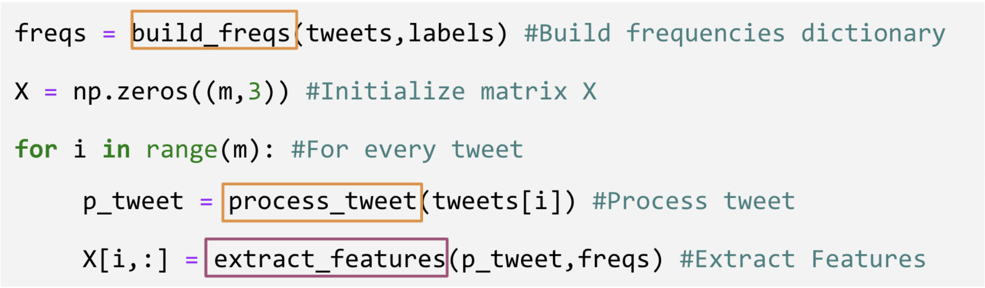
Over all, you start with a given text, you perform preprocessing, then you do feature extraction to convert text into numerical representation as follows:



Your X becomes of dimension $(m, 3)$ as follows.

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

When implementing it with code, it becomes as follows:



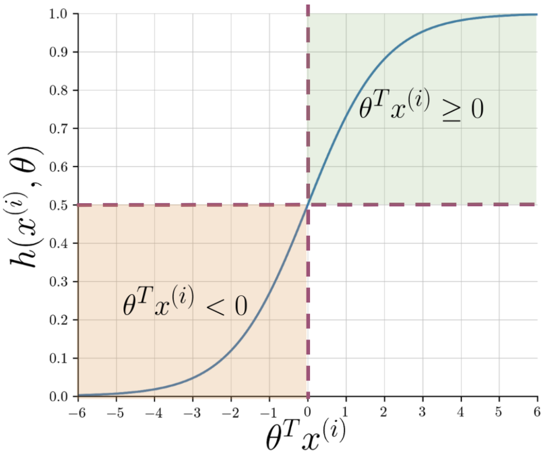
```
freqs = build_freqs(tweets,labels) #Build frequencies dictionary
X = np.zeros((m,3)) #Initialize matrix X
for i in range(m): #For every tweet
     p_tweet = process_tweet(tweets[i]) #Process tweet
     X[i,:] = extract_features(p_tweet,freqs) #Extract Features
```

You can see in the last step you are storing the extracted features as rows in your X matrix and you have m of these examples.

6. Logistic Regression Overview

Logistic regression makes use of the sigmoid function which outputs a probability between 0 and 1. The sigmoid function with some weight parameter θ and some input $x^{(i)}$ is defined as follows.

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



Note that as $\theta^T x^{(i)}$ gets closer and closer to $-\infty$ the denominator of the sigmoid function gets larger and larger and as a result, the sigmoid gets closer to 0 . On the other hand, as $\theta^T x^{(i)}$ gets closer and closer to ∞ the denominator of the sigmoid function gets closer to 1 and as a result the sigmoid also gets closer to 1.

Now given a tweet, you can transform it into a vector and run it through your sigmoid function to get a prediction as follows:

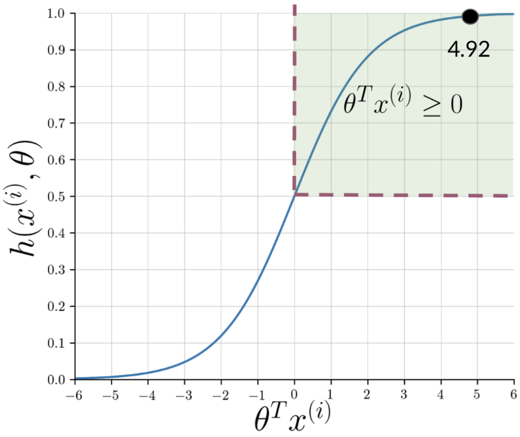
@YMourri and @AndrewYNg are tuning a GREAT AI model

↓

[tun, ai, great, model]

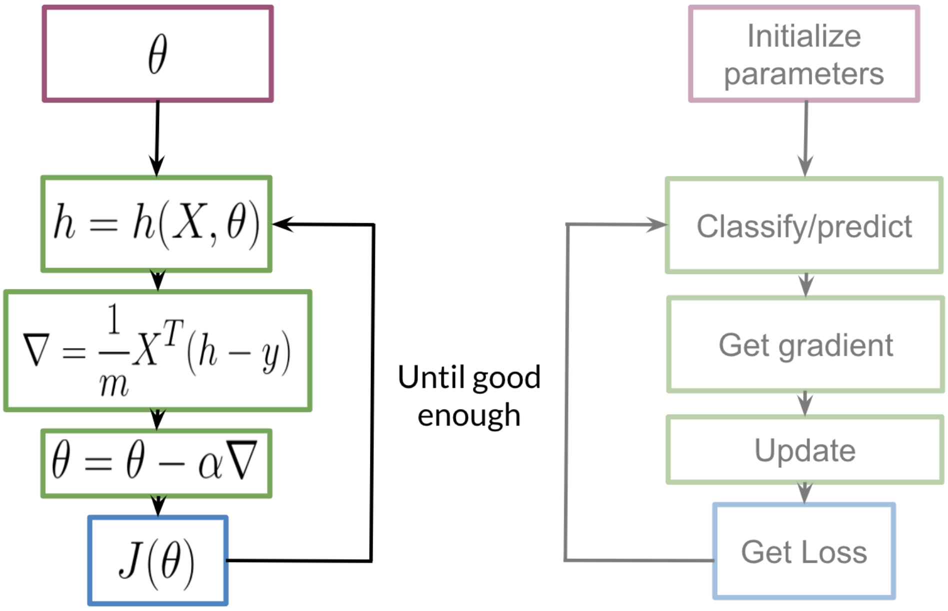
↓

$$x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix} \quad \theta = \begin{bmatrix} 0.00003 \\ 0.00150 \\ -0.00120 \end{bmatrix}$$



7. Logistic Regression: Training

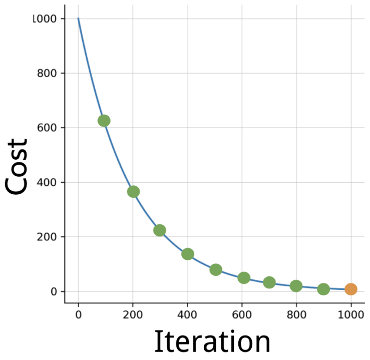
To train your logistic regression function, you will do the following:



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

Note: If you do not know what a gradient is, don't worry about it. I will show you what it is at then end of this week in an optional reading. In a nutshell, the gradient allows you to learn what θ is so that you can predict your tweet sentiment accurately.

Usually you keep training until the cost converges. If you were to plot the number of iterations versus the cost, you should see something like this:



8. Logistic Regression: Testing

To test your model, you would run a subset of your data, known as the validation set, on your model to get predictions. The predictions are the outputs of the sigmoid function. If the output is $\geq = 0.5$, you would assign it to a positive class. Otherwise, you would assign it to a negative class.

- $X_{val}\ Y_{val}\ \theta$

$h(X_{val}, \theta)$

$pred = h(X_{val}, \theta) \geq 0.5$

$$\begin{bmatrix} 0.3 \\ 0.8 \\ 0.5 \\ \vdots \\ h_m \end{bmatrix} \geq 0.5 = \begin{bmatrix} 0.3 \geq 0.5 \\ 0.8 \geq 0.5 \\ 0.5 \geq 0.5 \\ \vdots \\ pred_m \geq 0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix}$$

In the video, I briefly mentioned X validation. In reality, given your X data you would usually split it into three components. X_{train}, X_{val}, X_{test}. The distribution usually varies depending on the size of your data set. However, an 80, 10, 10 split usually works fine.

To compute accuracy, you solve the following equation:

$$\text{Accuracy} \longrightarrow \sum_{i=1}^{m} \frac{(pred^{(i)} == y_{val}^{(i)})}{m}$$

In other words, you go over all your training examples, m of them, and then for every prediction, if it was right you add a one. You then divide by m.