# NLP with Classification and Vector Spaces – Week1

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.
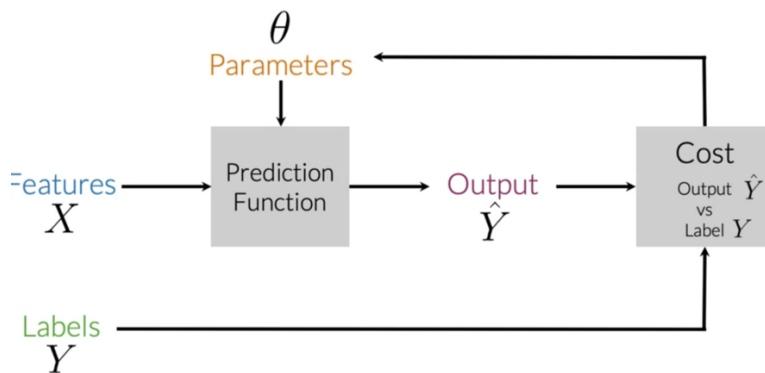

To perform sentiment analysis on a tweet, you first have to represent the text (i.e. "I am happy because I am learning NLP ") as features, you then train your logistic regression classifier, and then you can use it to classify the text.
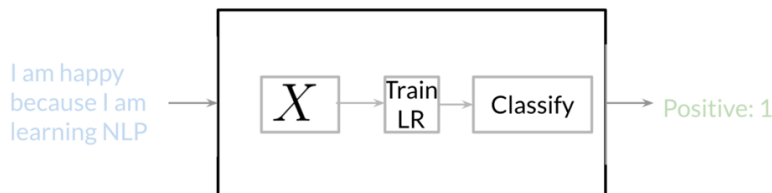
Note that in this case, you either classify 1, for a positive sentiment, or 0, for a negative sentiment.

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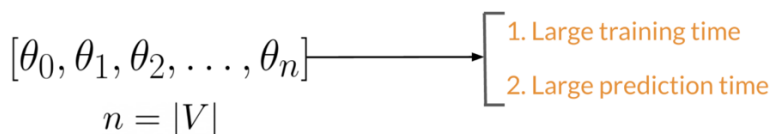
As you can see, as V gets larger, the vector becomes more sparse. Furthermore, we end up having many more features and end up training θ V parameters. This could result in larger training time, and large prediction time.

Feature Extraction with Frequencies

Given a corpus with positive and negative tweets as follows:

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

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

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

8

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

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

11

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

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.
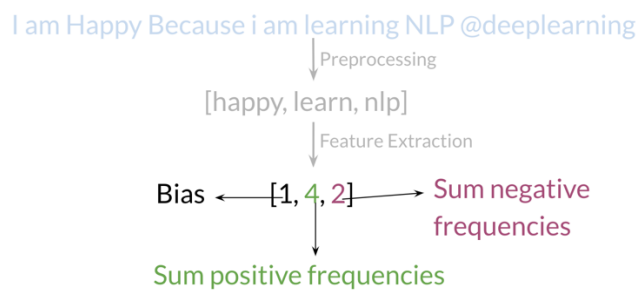
5. Convert all your words to lower case.

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

tuning GREAT AI model

tune
tun ← tuned
tuning

GREAT
Great → great
great

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

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

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:

I am Happy Because i am learning NLP @deeplearning

| Preprocessing

[happy, learn, nlp]

| Feature Extraction

Bias ⟵ [1, 4, 2] ⟶ Sum negative frequencies

Sum positive frequencies

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

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

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.
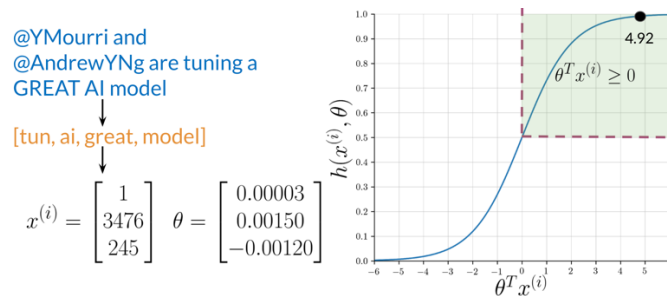
Logistic Regression Overview

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.
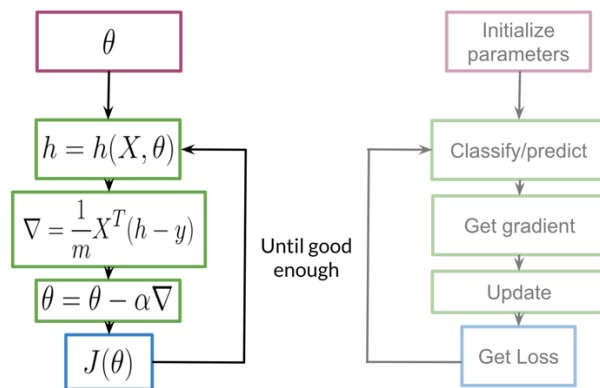
$$h(x^{(i)}, \theta) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$
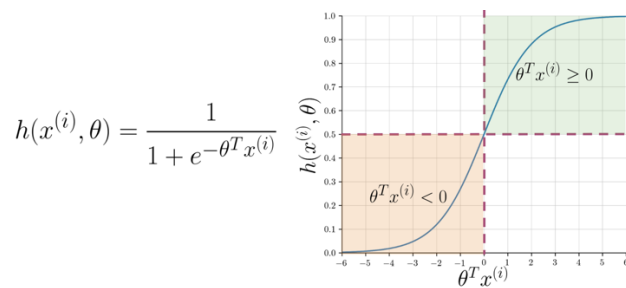
Note that as $\theta Tx(i)$ gets closer and closer to $-\infty$ the denominator of the sigmoid function gets larger and larger and as a result, the sigmoid gets closer to 00. On the other hand, as $\theta Tx(i)$ gets closer and closer to ∞ the denominator of the sigmoid function gets closer to 1 and 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]

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

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

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

Note: In a nutshell, the gradient allows you to learn what \theta$\theta$ is so that you can predict your tweet sentiment accurately.

5

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:
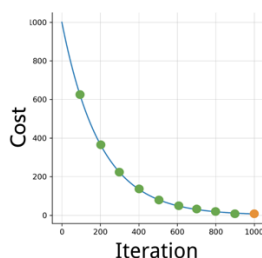
Logistic Regression: Testing

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$\geq$=0.5, you would assign it to a positive class. Otherwise, you would assign it to a negative class.

- X_{val} Y_{val} θ

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

$$pred = h(X_{val} \cdot \theta) \geq 0.5 \quad \begin{bmatrix} 0.3 \\ 0.8 \\ 0.5 \\ \vdots \\ h_m \end{bmatrix} \geq 0.5 = \begin{bmatrix} 0.3 \geq 0.5 \\ 0.8 \geq 0.5 \\ 0.5 > 0.5 \\ \vdots \\ pred_m \geq 0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix}$$

In reality, given your X data you would usually split it into three components. X_{train}, X_{val}, X_{test}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.

Optional Logistic Regression: Cost Function

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log h\left(x^{(i)}, \theta\right) + \left(1 - y^{(i)}\right) \log \left(1 - h\left(x^{(i)}, \theta\right)\right) \right]$$

---

1. I will tell you about the intuition behind why the cost function is designed the way it is. I will then show you how to derive the logistic regression cost function to get the gradients.

2. The logistic regression cost function is defined as

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log h(x^{(i)}, \theta) + (1 - y^{(i)}) \log(1 - h(x^{(i)}, \theta))]$$

As you can see in the picture above, if y=1 and you predict something close to 0, you get a cost close to ∞. The same applies for then $y=0$ and you predict something close to 1. On the other hand if you get a prediction equal to the label, you get a cost of 0. In either, case you are trying to minimize $J(\theta)$.

Math Derivation

To show you why the cost function is designed that way, let us take a step back and write up a function that compresses the two cases into one case.

$$P(y|x^{(i)}, \theta) = h(x^{(i)}, \theta)^{y^{(i)}} (1 - h(x^{(i)}, \theta)^{(1-y^{(i)})}$$

From the above, you can see that when y = 1$y=1$, you get h(x^{(i)}, \theta)$h(x(i),\theta)$, and when y = 0$y=0$, you get (1-h(x^{(i)}, \theta))$(1-h(x(i),\theta))$, which makes sense, since the two probabilities equal to 1. In either case, you want to maximize the function h(x^{(i)}, \theta)$h(x(i),\theta)$ by making it as close to 11 as possible. When y=0$y=0$, you want (1-h(x^{(i)}, \theta))$(1-h(x(i),\theta))$ to be 00, and therefore h(x^{(i)}, \theta)$h(x(i),\theta)$ close to 11. When y=1$y=1$, you want h(x^{(i)}, \theta)=1$h(x(i),\theta)=1$.

Now we want to find a way to model the entire data set and not just one example. To do so, we will define the likelihood as follows:

L(\theta) = \prod_{i=1}^m h(\theta, x^{(i)})^{y^{(i)}} (1-h(\theta, x^{(i)}))^{(1-y^{(i)})}$L(\theta)=\prod i=1mh(\theta,x(i))y(i)(1-h(\theta,x(i)))(1-y(i))$

The \prod$\prod$ symbol tells you that you are multiplying the terms together and not adding them. Note that if we mess up the classification of one example, we end up messing up the overall likelihood score, which is exactly what we intended. We want to fit a model to the entire dataset where all data points are related. One issue is that as mm gets larger, what happens to L(\theta)$L(\theta)$? It goes close to zero, because both numbers h(x^{(i)}, \theta)$h(x(i),\theta)$ and (1-h(x^{(i)}, \theta))$(1-h(x(i),\theta))$ are bounded between 00 and 11. Since we are trying to maximize h(\theta, x^{(i)})$h(\theta,x(i))$ in L(\theta)$L(\theta)$, we can introduce the log and just maximize the log of the function. (We are maximizing the same function just in a different space). Introducing the \log$\log$, allows us to

write the \loglog of a product as the sum of each \loglog. Here are two identities that will come in handy:

\log a*b*c = \log a + \log b + \log c $\log a*b*c = \log a + \log b + \log c$

\log a^b = b \log a $\log a^b = b \log a$

Given the two identities above, we can rewrite the equation as follows:

$$\max_{h(x^{(i)},\theta)} \log L(\theta) = \log \prod_{i=1}^{m} h(x^{(i)},\theta)^{y^{(i)}} (1-h(x^{(i)},\theta)^{(1-y^{(i)})}$$

$$= \sum_{i=1}^{m} \log h(x^{(i)},\theta)^{y^{(i)}} (1-h(x^{(i)},\theta)^{(1-y^{(i)})}$$

$$= \sum_{i=1}^{m} \log h(x^{(i)},\theta)^{y^{(i)}} + \log(1-h(x^{(i)},\theta)^{(1-y^{(i)})}$$

$$= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)},\theta) + (1-y^{(i)}) \log(1-h(x^{(i)},\theta))$$

Hence, we now divide by m, because we want to see the average cost.

$$\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)},\theta) + (1-y^{(i)}) \log(1-h(x^{(i)},\theta))$$

Remember that we were maximizing h(\theta, x^{(i)}) $h(\theta,x_{(i)})$ in the equation above. It turns out that maximizing an equation is the same as minimizing its negative. Think of x^2 x_2, feel free to plot it to see that for you yourself. Hence we add a negative sign and we end up minimizing the cost function as follows.

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log h\left(x^{(i)},\theta\right) + \left(1-y^{(i)}\right) \log \left(1-h\left(x^{(i)},\theta\right)\right) \right]$$

A vectorized implementation is:

$$h = g(X\theta)$$
$$J(\theta) = \frac{1}{m} \cdot \left(-y^T \log(h) - (1-y)^T \log(1-h)\right)$$

Logistic Regression: Gradient

This is an optional reading where I explain gradient descent in more detail. Remember, previously I gave you the gradient update step, but did not explicitly explain what is going on behind the scenes.

The general form of gradient descent is defined as:

> Repeat$\{\theta_j:=\theta_j-\alpha\partial\partial\theta_jJ(\theta)\}$

For all **j**j. We can work out the derivative part using calculus to get:

> Repeat$\{\theta_j:=\theta_j-\alpha m\sum^{i=1m}(h(x(i),\theta)-y(i))x(i)j\}$

A vectorized implementation is:

\theta := \theta - \frac{\alpha}{m} X^{T} (H(X , \theta) - Y)$\theta:=\theta-\frac{m}{\alpha}X_T(H(X,\theta)-Y)$

Partial derivative of J(θ)

First calculate derivative of sigmoid function (it will be useful while finding partial derivative of J(θ)):

$$
\begin{aligned}
h(x)' &= \left(\frac{1}{1+e^{-x}}\right)' = \frac{-(1+e^{-x})'}{(1+e^{-x})^2} = \frac{-1'-(e^{-x})'}{(1+e^{-x})^2} = \frac{0-(-x)'(e^{-x})}{(1+e^{-x})^2} = \frac{-(-1)(e^{-x})}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{e^{-x}}{1+e^{-x}}\right) = h(x)\left(\frac{+1-1+e^{-x}}{1+e^{-x}}\right) = h(x)\left(\frac{1+e^{-x}}{1+e^{-x}}-\frac{1}{1+e^{-x}}\right) = h(x)(1-h(x))
\end{aligned}
$$

 Note that we computed the partial derivative of the sigmoid function. If we were to derive h(x^{(i)}, \theta)$h(x(i),\theta)$ with respect to \theta_jθ_j, you would get h(x^{(i)}, \theta)(1-h(x^{(i)}, \theta))x^{(i)}_j$h(x(i),\theta)(1-h(x(i),\theta))x_j(i)$. Note that we used the chain rule there, because we multiply by the derivative of \theta^Tx^{(i)}$\theta_Tx(i)$ with respect to \theta_jθ_j. Now we are ready to

find out resulting partial derivative:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} log(h(x^{(i)}, \theta)) + (1 - y^{(i)}) log(1 - h(x^{(i)}, \theta)) \right]$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\partial}{\partial \theta_j} log(h(x^{(i)}, \theta)) + (1 - y^{(i)}) \frac{\partial}{\partial \theta_j} log(1 - h(x^{(i)}, \theta)) \right]$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[\frac{y^{(i)} \frac{\partial}{\partial \theta_j} h(x^{(i)}, \theta)}{h(x^{(i)}, \theta)} + \frac{(1 - y^{(i)}) \frac{\partial}{\partial \theta_j} (1 - h(x^{(i)}, \theta))}{1 - h(x^{(i)}, \theta)} \right]$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[\frac{y^{(i)} \frac{\partial}{\partial \theta_j} h(x^{(i)}, \theta)}{h(x^{(i)}, \theta)} + \frac{(1 - y^{(i)}) \frac{\partial}{\partial \theta_j} (1 - h(x^{(i)}, \theta))}{1 - h(x^{(i)}, \theta)} \right]$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[\frac{y^{(i)} h(x^{(i)}, \theta)(1 - h(x^{(i)}, \theta)) \frac{\partial}{\partial \theta_j} \theta^T x^{(i)}}{h(x^{(i)}, \theta)} + \frac{-(1 - y^{(i)}) h(x^{(i)}, \theta)(1 - h(x^{(i)}, \theta)) \frac{\partial}{\partial \theta_j} \theta^T x^{(i)}}{1 - h(x^{(i)}, \theta)} \right]$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[\frac{y^{(i)} h(x^{(i)}, \theta)(1 - h(x^{(i)}, \theta)) \frac{\partial}{\partial \theta_j} \theta^T x^{(i)}}{h(x^{(i)}, \theta)} - \frac{(1 - y^{(i)}) h(x^{(i)}, \theta)(1 - h(x^{(i)}, \theta)) \frac{\partial}{\partial \theta_j} \theta^T x^{(i)}}{1 - h(x^{(i)}, \theta))} \right]$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} (1 - h(x^{(i)}, \theta)) x_j^{(i)} - (1 - y^{(i)}) h(x^{(i)}, \theta) x_j^{(i)} \right]$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} (1 - h(x^{(i)}, \theta)) - (1 - y^{(i)}) h(x^{(i)}, \theta) \right] x_j^{(i)}$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} - y^{(i)} h(x^{(i)}, \theta) - h(x^{(i)}, \theta) + y^{(i)} h(x^{(i)}, \theta) \right] x_j^{(i)}$$

$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} - h(x^{(i)}, \theta) \right] x_j^{(i)}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left[h(x^{(i)}, \theta) - y^{(i)} \right] x_j^{(i)}$$

The vectorized version:

$$\nabla J(\theta) = \frac{1}{m} \cdot X^T \cdot (H(X, \theta) - Y)$$

How to refresh your workspace

There may come a time when you want to refresh your workspace to get the most updated version of the starter code. This may be the case if you wish to start over, or if the deeplearning.ai team has provided an update with bug fixes or enhancements. To refresh your workspace, please follow these steps:

- In any Jupyter notebook, first, save your work by going clicking File -> Download as -> Notebook (.ipynb)

- Next, click File -> Open. This opens up the file directory.

- Select the notebook you wish to refresh from the list by clicking the checkbox next to the filename

- Click the trashcan icon at the top to delete the notebook

- Copy this text: **?forceRefresh=true** and paste it onto the end of the URL in your browser bar then hit <Enter>.

- You will see your workspace refresh with the updated copy of the notebook