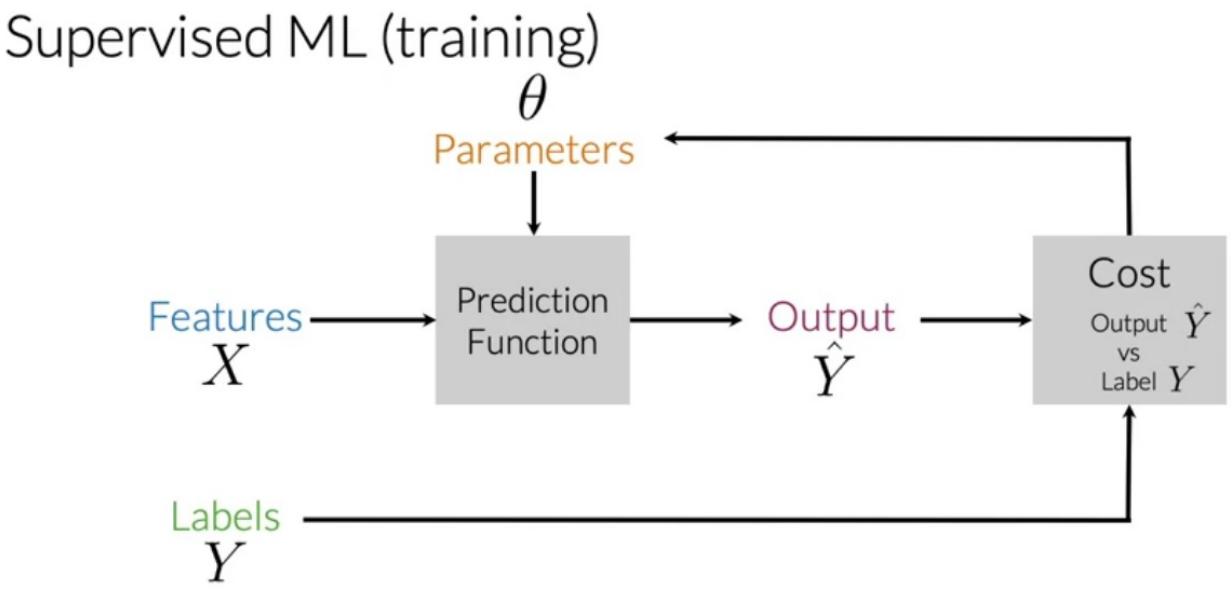
Natural Language Processing with Classification and Vector Spaces || Week — 01 (Logistic regression)

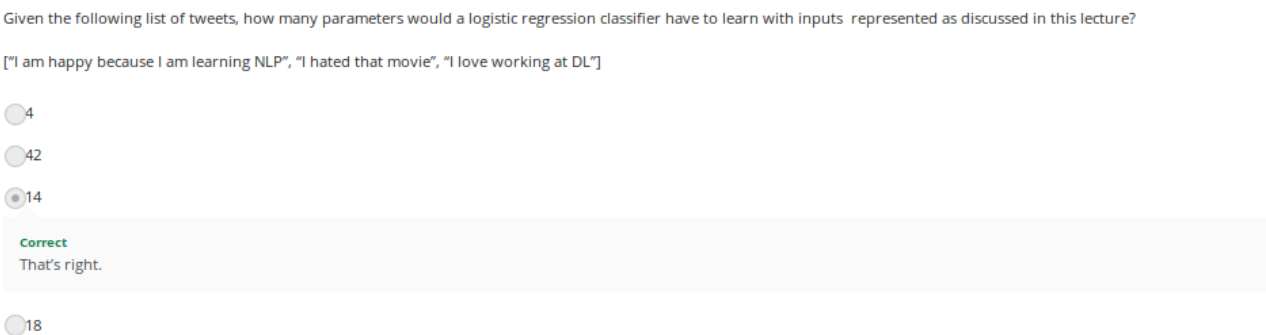
medium.com/@aakashgoel12/natural-language-processing-with-classification-and-vector-spaces-week-01-logistic-4a633230f8d2

July 21, 2020



Logistic Regression — Supervised Learning

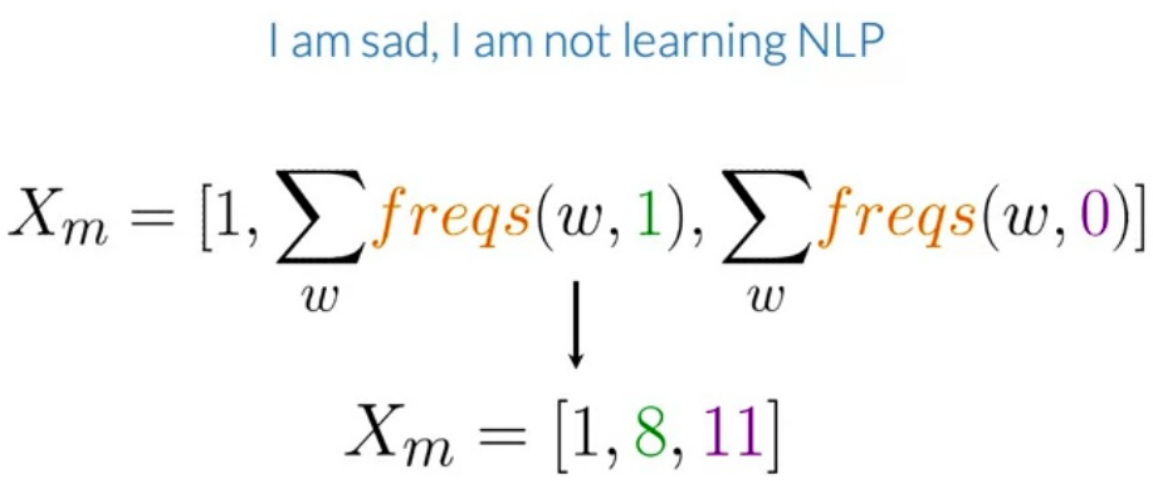
- Extract Features
- Train your model
- Classify tweet



13 unique words and 1 for bias

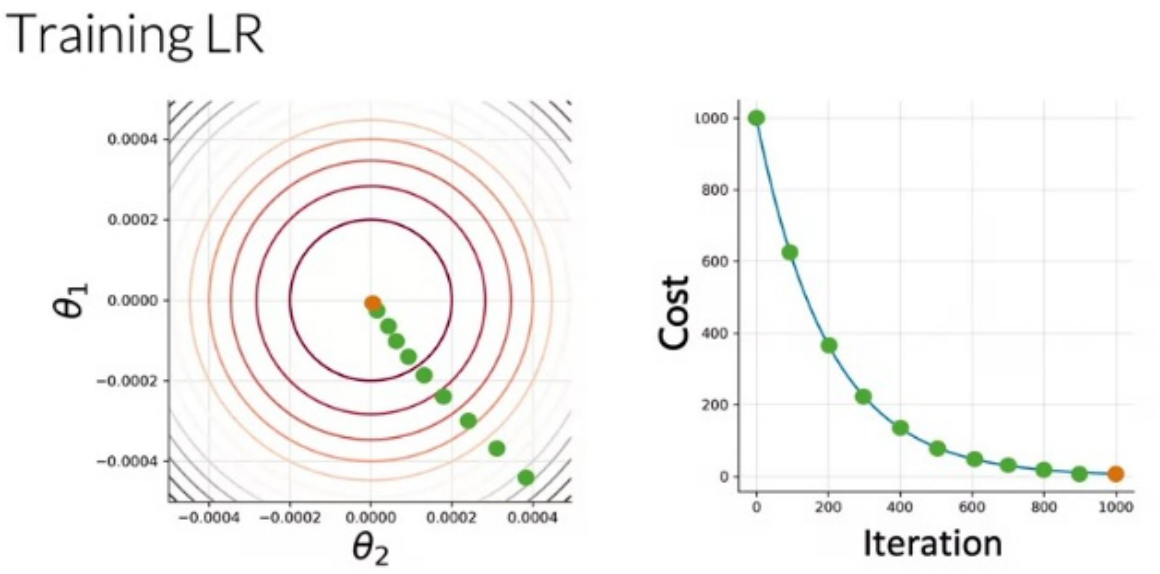
How to represent tweet ?

- Make vocabulary of size V by finding unique words in tweet. Tweet will be represented by binary vectorizer (0 or 1) of size V. So, logistic regression will learn V+1 parameters and that will be too large depend on vocab size.
- Represent tweet by 3 features [1 (bias), Sum of frequency of positive tweet, Sum of frequency of negative tweet]. For each word in vocab, count how may times it appeared for positive and negative tweet.



Representation of tweet

- Refer Jupyter NB on Tweet Pre-processing
- Refer Jupyter NB on Tweet sentiment Analysis using Logistic Regression



Logistic Regression Training Process

Logistic regression: regression and a sigmoid

Logistic regression takes a regular linear regression, and applies a sigmoid to the output of the linear regression.

Regression:

$$z = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots \theta_N x_N$$

Note that the θ values are "weights". If you took the Deep Learning Specialization, we referred to the weights with the `w` vector. In this course, we're using a different variable θ to refer to the weights.

Logistic regression

$$h(z) = \frac{1}{1 + \exp^{-z}}$$
$$z = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots \theta_N x_N$$

We will refer to 'z' as the 'logits'.

About Logistic Regression

Part 1.2 Cost function and Gradient

The cost function used for logistic regression is the average of the log loss across all training examples:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(h(z(\theta)^{(i)})) + (1 - y^{(i)}) \log(1 - h(z(\theta)^{(i)})) \tag{5}$$

- m is the number of training examples
- $y^{(i)}$ is the actual label of the i-th training example.
- $h(z(\theta)^{(i)})$ is the model's prediction for the i-th training example.

The loss function for a single training example is

$$Loss = -1 \times \left(y^{(i)} \log(h(z(\theta)^{(i)})) + (1 - y^{(i)}) \log(1 - h(z(\theta)^{(i)})) \right)$$

- All the h values are between 0 and 1, so the logs will be negative. That is the reason for the factor of -1 applied to the sum of the two loss terms.
- Note that when the model predicts 1 ($h(z(\theta)) = 1$) and the label y is also 1, the loss for that training example is 0.
- Similarly, when the model predicts 0 ($h(z(\theta)) = 0$) and the actual label is also 0, the loss for that training example is 0.
- However, when the model prediction is close to 1 ($h(z(\theta)) = 0.9999$) and the label is 0, the second term of the log loss becomes a large negative number, which is then multiplied by the overall factor of -1 to convert it to a positive loss value. $-1 \times (1 - 0) \times log(1 - 0.9999) \approx 9.2$ The closer the model prediction gets to 1, the larger the loss.

Logistic — Cost Function

Update the weights

To update your weight vector θ, you will apply gradient descent to iteratively improve your model's predictions.
The gradient of the cost function J with respect to one of the weights θ_j is:

$$\nabla_{\theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h^{(i)} - y^{(i)}) x_j \tag{5}$$

- 'i' is the index across all 'm' training examples.
- 'j' is the index of the weight θ_j, so x_j is the feature associated with weight θ_j
- To update the weight θ_j, we adjust it by subtracting a fraction of the gradient determined by α:
$$\theta_j = \theta_j - \alpha \times \nabla_{\theta_j} J(\theta)$$
- The learning rate α is a value that we choose to control how big a single update will be.

Weight Update

Instructions: Implement gradient descent function

- The number of iterations `num_iters` is the number of times that you'll use the entire training set.
- For each iteration, you'll calculate the cost function using all training examples (there are `m` training examples), and for all features.
- Instead of updating a single weight θ_i at a time, we can update all the weights in the column vector:

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{pmatrix}$$

- θ has dimensions (n+1, 1), where 'n' is the number of features, and there is one more element for the bias term θ_0 (note that the corresponding feature value $\mathbf{x_0}$ is 1).
- The 'logits', 'z', are calculated by multiplying the feature matrix 'x' with the weight vector 'theta'. $z = \mathbf{x}\theta$
  - $\mathbf{x}$ has dimensions (m, n+1)
  - θ: has dimensions (n+1, 1)
  - $\mathbf{z}$: has dimensions (m, 1)
- The prediction 'h', is calculated by applying the sigmoid to each element in 'z': $h(z) = sigmoid(z)$, and has dimensions (m,1).
- The cost function J is calculated by taking the dot product of the vectors 'y' and 'log(h)'. Since both 'y' and 'h' are column vectors (m,1), transpose the vector to the left, so that matrix multiplication of a row vector with column vector performs the dot product.

$$J = \frac{-1}{m} \times \left(\mathbf{y}^T \cdot log(\mathbf{h}) + (\mathbf{1} - \mathbf{y})^T \cdot log(\mathbf{1} - \mathbf{h})\right)$$

- The update of theta is also vectorized. Because the dimensions of $\mathbf{x}$ are (m, n+1), and both $\mathbf{h}$ and $\mathbf{y}$ are (m, 1), we need to transpose the $\mathbf{x}$ and place it on the left in order to perform matrix multiplication, which then yields the (n+1, 1) answer we need:

$$\theta = \theta - \frac{\alpha}{m} \times \left(\mathbf{x}^T \cdot (\mathbf{h} - \mathbf{y})\right)$$

Gradient Descent