Fake News Detection With Logistic Regression

Member 1: باسم مجدي محمود محمد

ID: 20170147

Member 2: ادهم طارق كمال سعيد

ID: 20170079

Member 3: اسلام هشام سيد

ID: 20170094

Member 4: احمد اسماعيل محمد ابو السعود

ID: 20170011

Member 5: اياد تامر عبد الحميد

ID: 20170125

1) Problem description:

Fake news has become a problem of great impact in our information driven society because of the continuous and intense fakesters content distribution. Information quality in news feeds is under questionable veracity calling for automated tools to detect fake news articles. Due to many faces of fakesters, creating such tool is a challenging problem. In this work, we propose a model for fake news detection using content based features and Machine Learning (ML) algorithms. To conclude in most accurate model, we evaluate logistic regression.

2) Model design:

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, it also estimates the parameters of a logistic model *(a form of binary regression)*. Mathematically, a binary logistic model has a dependent variable with two possible values, where the two values are labeled "0" as false and "1" as true.

It's used to model the probability of a certain class or event existing such as (pass/fail, win/lose, fake/true).

The sigmoid function is defined as:

$$h(z) = \frac{1}{1 + \exp^{-z}}$$

It maps the input 'z' to a value that ranges between 0 and 1, and so it can be treated as a probability.

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'.

Cost Functions and Gradient:

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

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

- 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.

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 θj is:

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

- 'i' is the index across all 'm' training examples.
- 'j' is the index of the weight θj, so xj is the feature associated with weight θj
- To update the weight θj, 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 $\alpha\alpha$ is a value that we choose to control how big a single update will be.

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 θi 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 $\theta 0$ (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': $\hbar(z)=sigmoid(z)$, 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:

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

Extracting the features:

- Given a list of tweets, extract the features and store them in a matrix. You will extract two features.
    - The first feature is the number of positive words in a tweet.
    - The second feature is the number of negative words in a tweet.
- Then train your logistic regression classifier on these features.
- Test the classifier on a validation set.

Implement the extract features function:

- This function takes in a single tweet.
- Process the tweet using the imported process_tweet() function and save the list of tweet words.
- Loop through each word in the list of processed words
    - For each word, check the freqs dictionary for the count when that word has a positive '1' label. (Check for the key (word, 1.0)
    - Do the same for the count for when the word is associated with the negative label '0'. (Check for the key (word, 0.0).)

Training The Model:

To train the model:

- Stack the features for all training examples into a matrix X.
- Call gradientDescent, which you've implemented above.

Test your logistic regression:

Predict whether a tweet is positive or negative.

- Given a tweet, process it, then extract the features.
- Apply the model's learned weights on the features to get the logits.
- Apply the sigmoid to the logits to get the prediction (a value between 0 and 1).

$$y_{pred} = sigmoid(\mathbf{x} \cdot \theta)$$

Check performance using the test set:

After training the model using the training set above, check how your model might perform on real, unseen data, by testing it against the test set.

Implement test_logistic_regression

- Given the test data and the weights of your trained model, calculate the accuracy of your logistic regression model.

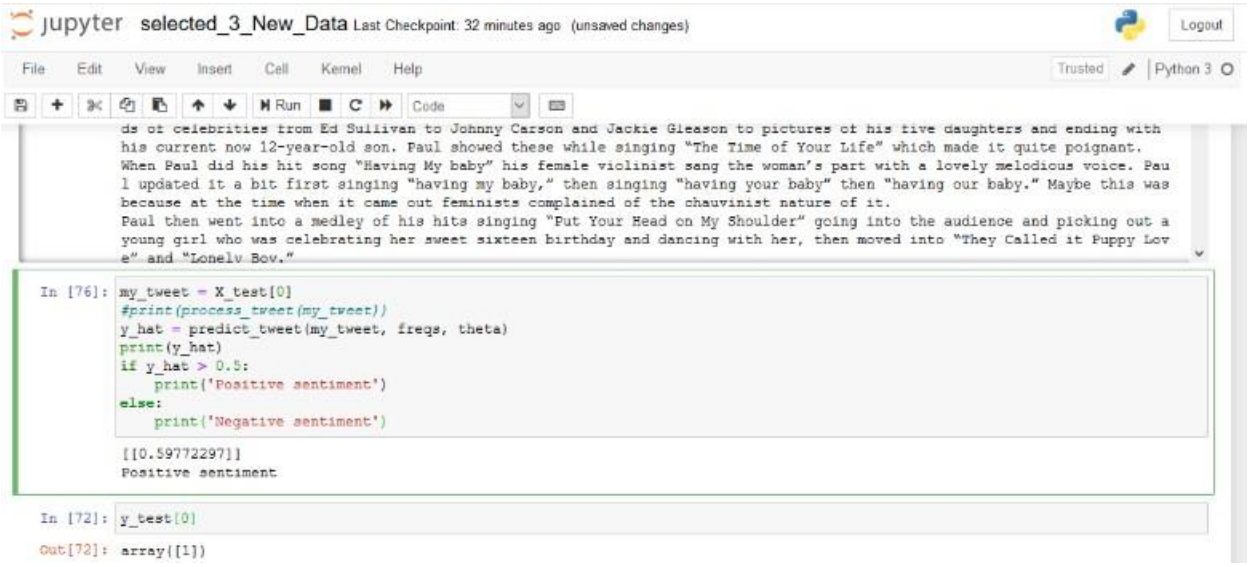
- Use your predict_tweet() function to make predictions on each tweet in the test set.
- If the prediction is > 0.5, set the model's classification y_hat to 1, otherwise set the model's classification y_hat to 0.
- A prediction is accurate when y_hat equals test_y. Sum up all the instances when they are equal and divide by m .

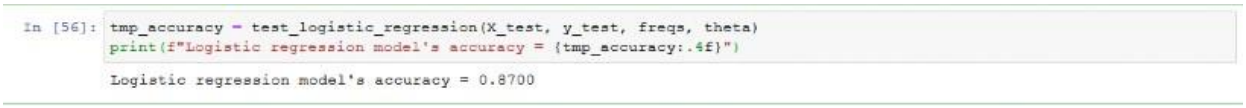
Error Analysis:

In this phase a detection of misclassified news is occurred.

3) Experimental results:

Test the functionality of the model by figuring out if the news was actually predicted as true or not, in this case the model figured out that the news was true as it was in the data set



4) Model performance:

The accuracy of the model: