Drishte Assignment Task Submitted by Yashi Kesarwani

Reviews Classification using Neural Network

Implementation of Multi-Layer Perceptron (MLP)

Goal - The Goal of this assignment is to implement a multi-layer perceptron from scratch.

Objective- The objective of this project is to classify the reviews given by users as positive or negative.

About Dataset

Reviews.txt is the dataset of the collections of reviews given by different users. **Label.txt** is the dataset of the corresponding results of the same row of the reviews.txt dataset i.e. POSITIVE or NEGATIVE.

The data in reviews.txt we're using has already been preprocessed a bit and contains only lower-case characters. That's why we treat different variations of the same word, like The, the, and THE, all the same way.

Dimension of the dataset: Length of reviews and label dataset is 25000. Each row of label dataset gives result to the corresponding review result of reviews dataset.

Algorithm or Approach

1-We create three Counter objects, one for words from positive reviews, one for words from negative reviews, and one for all the words.

2-For each word in a positive review, increase the count for that word in both our positive counter and the total words counter; likewise, for each word in a negative review, increase the count for that word in both our negative counter and the total words counter.

We do listing the words used in positive reviews and negative reviews, respectively, ordered from most to least commonly used.

3-As we can see, common words like "the" appear very often in both positive and negative reviews. Instead of finding the most common words in positive or negative reviews, what we really want are the words found in positive reviews more often than in negative reviews, and vice versa. To accomplish this, we do calculate the **ratios** of word usage between positive and negative reviews.

4-Checking all the words we've seen and calculate the ratio of positive to negative uses and store that ratio in pos_neg_ratios.

5-The positive-to-negative ratio for a given word is calculated with **positive_counts[word] / float(negative_counts[word]+1)**. Notice the +1 in the denominator – that ensures we don't divide by zero for words that are only seen in positive reviews.

When we observe at the values just calculated, we see the following:

- Words that we would expect to see more often in positive reviews – like "amazing" – have a ratio greater than 1. The more skewed a word is toward positive, the farther from 1 its positive-to-negative ratio will be.
- Words that we would expect to see more often in negative reviews – like "terrible" – have positive values that are less than 1. The more skewed a word is toward negative, the closer to zero its positive-to-negative ratio will be.
- Neutral words, which don't really convey any sentiment because we would expect to see them in all sorts of reviews – like "the" – have values very close to 1. A perfectly neutral word – one that was used in exactly the same number of positive reviews as negative reviews – would be almost exactly 1. The +1 we suggested you add to the denominator slightly biases words toward negative, but it won't matter because it will be a tiny bias and later, we'll be ignoring words that are too close to neutral anyway.

Ok, the ratios tell us which words are used more often in positive or negative reviews, but the specific values we've calculated are a bit difficult to work with. A very positive word like "amazing" has a value above 4, whereas a very negative word like "terrible" has a value around 0.18. Those values aren't easy to compare for a couple of reasons:

- Right now, 1 is considered neutral, but the absolute value of the positive-to-negative ratios of very positive words is larger than the absolute value of the ratios for the very negative words. So, there is no way to directly compare two numbers and see if one word conveys the same magnitude of positive sentiment as another word conveys negative sentiment. So, we should center all the values around neutral so the absolute value from neutral of the postive-to-negative ratio for a word would indicate how much sentiment (positive or negative) that word conveys.
- When comparing absolute values it's easier to do that around zero than one.

To fix these issues, we'll convert all of our ratios to new values using logarithms.

Convert all the ratios to logarithms. (i.e. use np.log(ratio))

In the end, extremely positive and extremely negative words will have positive-to-negative ratios with similar magnitudes but opposite signs.

NOTE:

- For any positive words, convert the ratio using np.log(ratio)
- For any negative words, convert the ratio using -np.log(1/(ratio + 0.01))

The results are extremely positive and extremely negative words having positive-to-negative ratios with similar magnitudes but opposite signs.

If everything worked, now you should see neutral words with values close to zero. In this case, "the" is near zero but slightly positive, so it was probably used in more positive reviews than negative reviews. But look at "amazing"s ratio - it's above 1, showing it is clearly a word with positive sentiment. And "terrible" has a similar score, but in the opposite direction, so it's below -1. It's now clear that both of these words are associated with specific, opposing sentiments.

We first display all the words by **pos_neg_ratios.most_common()**, ordered by how associated they are with positive reviews.

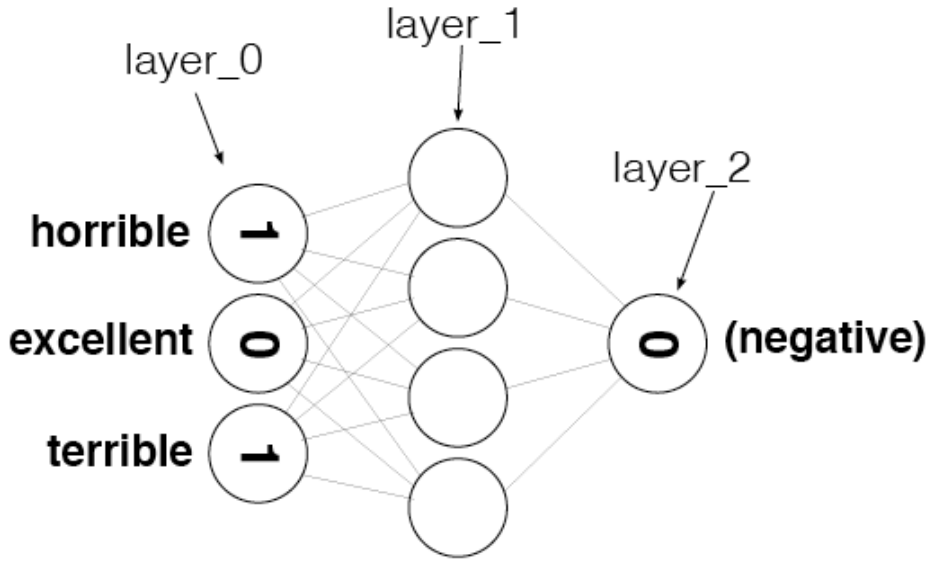
Then afterwards we display the 30 words most associated with negative reviews by reversing the order of the first list and then looking at the first 30 words.**reversed(pos_neg_ratios.most_common())[0:30]**

We should continue to see values similar to the earlier ones we checked – neutral words will be close to 0, words will get more positive as their ratios approach and go above 1, and words will get more negative as their ratios approach and go below -1. That's why we decided to use the logs instead of the raw ratios.

Creating the Input/Output Data

Create a set named vocab that contains every word in the vocabulary. The length of this vocab is 74074.

Take a look at the following image. It represents the layers of the neural network we'll be building throughout this project. layer_0 is the input layer, layer_1 is a hidden layer, and layer_2 is the output layer.



Creating a numpy array called `layer_0` and initialize it to all zeros. We create `layer_0` as a 2-dimensional matrix with 1 row and `vocab_size` columns.

layer_0 contains one entry for every word in the vocabulary. We need to make sure we know the index of each word and create a lookup table that stores the index of every word. For this, we Create a dictionary of words as **word2index** in the vocabulary mapped to index positions to be used in layer_0 and display the map of words to indices

update_input_layer is counting how many times each word is used in the given review, and then store those counts at the appropriate indices inside layer_0. Modifying the global layer_0 to represent the vector form of review. The element at a given index of layer_0 should represent how many times the given word occurs in the review.

get_target_for_labels is returning 0 or 1, depending on whether the given label is NEGATIVE or POSITIVE, respectively.

Building a Neural Network

Create a Sentiment Network with the given parameters

reviews(list) - List of reviews used for training, **labels(list)** - List of POSITIVE/NEGATIVE labels associated with the given reviews, **hidden_nodes(int)** - Number of nodes to create in the hidden layer, **learning_rate(float)** - Learning rate to use while training

We do some data preprocessing using the function **pre_process_data(self, reviews, labels)**

Initialize the network using **init_network**(**self, input_nodes, hidden_nodes, output_nodes,learning_rate)**

Then we do training of the model. We split out our dataset into training and testing set. We test the network's performance against the last 1000 reviews (the ones we held out from our training set).

Learning rate = 0.1, the training accuracy is about 50%.

Learning rate = 0.01, the training accuracy is about 50%. (Not much effect)

Learning rate = 0.001, the training accuracy is about 62%.

With a learning rate of 0.001, the network finally improved during training. It's still not very good, but it shows that this solution has potential. We will try improve it using some optimization techniques.

Detecting Neural Noise

Some word occurs rapidly so their weight is high and hence they affect more to the hidden layer unnecessarily. So, in the next step we try to reduce these irrelevant noise by making them not counting the no. of times they occur but only check whether a word is used or not.

There are many words like ‘the’, ‘it’, ‘on’, ‘ ’, ‘.’, etc. Which does not make any effect in deciding the label to the reviews. These are called the noise. So, in the further steps we will try to

reduce or eliminate inefficiencies by adding two parameters called **“polarity cut-off”** and **“min-count”**.

Reducing Noise in Our Input Data

- Modifying **update_input_layer** so it does not count how many times each word is used, but rather just stores whether or not a word was used.
- **Training Accuracy = 83.6% and testing accuracy = 85 %**

Making our Network More Efficient

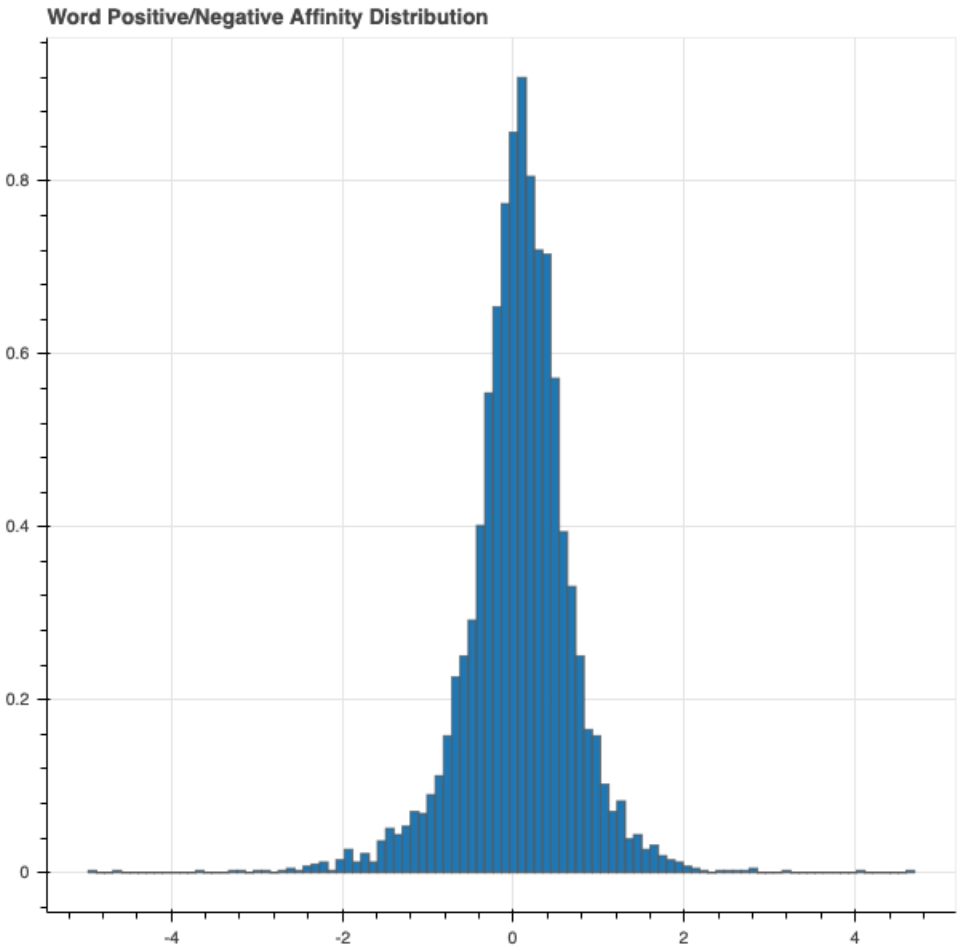
Increasing the inefficiency by eliminating unnecessary multiplications and additions that occur during forward and backward propagation.

- Removing the update_input_layer function - we will not need it in this version.
- Modifying init_network:
    - We no longer need a separate input layer, so we remove it
    - We will be dealing with the old hidden layer more directly, so we create self.layer_1, a two-dimensional matrix with shape 1 x hidden_nodes, with all values initialized to zero
- Modifying train:
    - Change the name of the input parameter training_reviews to training_reviews_raw.
    - At the beginning of the function, we want to preprocess reviews to convert them to a list of indices (from word2index) that are actually used in the review. These lists should contain the indices for words found in the review.
    - Remove call to update_input_layer
    - Use self's layer_1 instead of a local layer_1 object.
    - In the forward pass, replace the code that updates layer_1 with new logic that only adds the weights for the indices used in the review.
    - When updating weights_0_1, only update the individual weights that were used in the forward pass.
- Modify run:
    - Remove call to update_input_layer
    - Use self's layer_1 instead of a local layer_1 object.
    - Much like we did in train, we will need to pre-process the review so we can work with word indices, then update layer_1 by adding weights for the indices used in the review.

Training Accuracy = 84%, Testing Accuracy = 85.5%

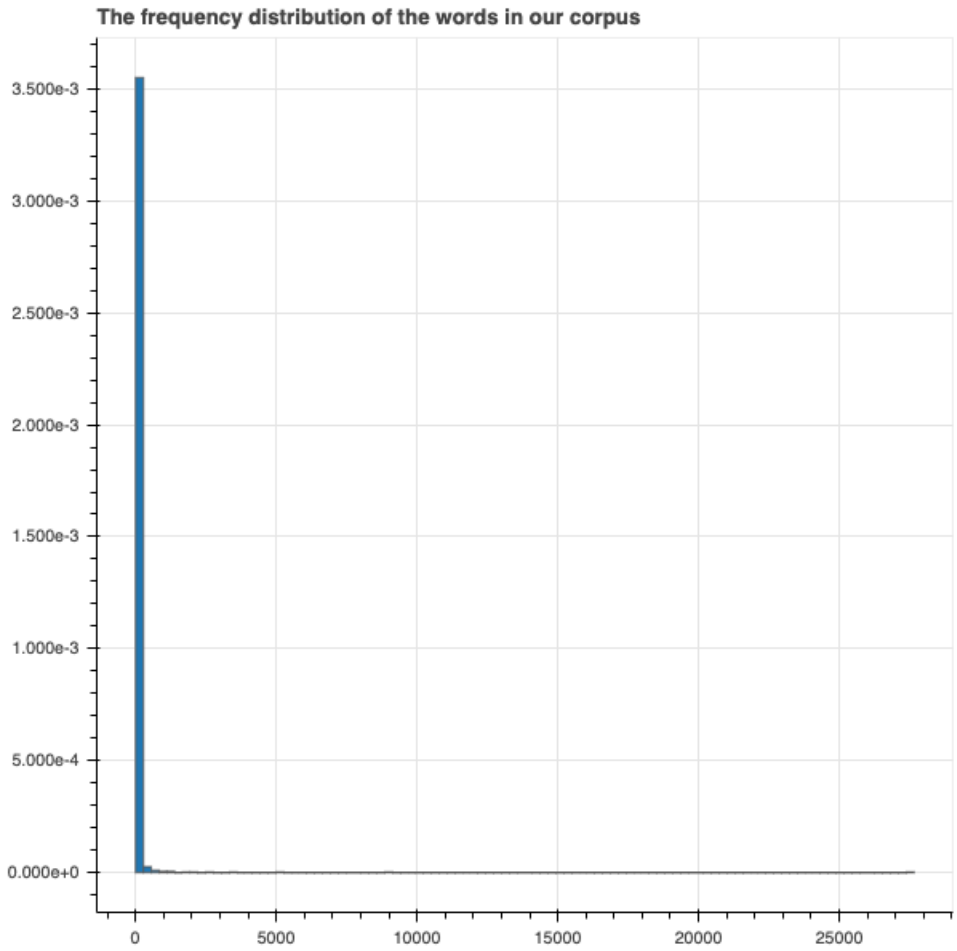
Further Noise Reduction

Graph-1



This histogram represents the postive to negative ratios of different word. So, those which are at the centre and at the highest peak have almost neutral effect and hence they can cutted down using the parameter called **“polarity_cutoff"** from our training review vocab list so as to increase the training accuracy.

Graph-2



This is the graph showing the frequency distirbutuion of the words. Those which are occuring only once or very few times cannot provide an effective co-relation hence they are irrelevant to us. So, we use a parameter called **"min_count"** to eliminate these irrelevant words.

Reducing Noise by Strategically Reducing the Vocabulary and Improve SentimentNetwork'sperformance by reducing more noise in the vocabulary by doing the following:

- Modify pre_process_data in SentimentNetwork class:
    - Add two additional parameters: min_count and polarity_cutoff
    - Calculate the positive-to-negative ratios of words used in the reviews.
    - so words are only added to the vocabulary if they occur in the vocabulary more than min_count times.
    - so words are only added to the vocabulary if the absolute value of theirpostive-to-negative ratio is at least polarity_cutoff
- Modify __init__:

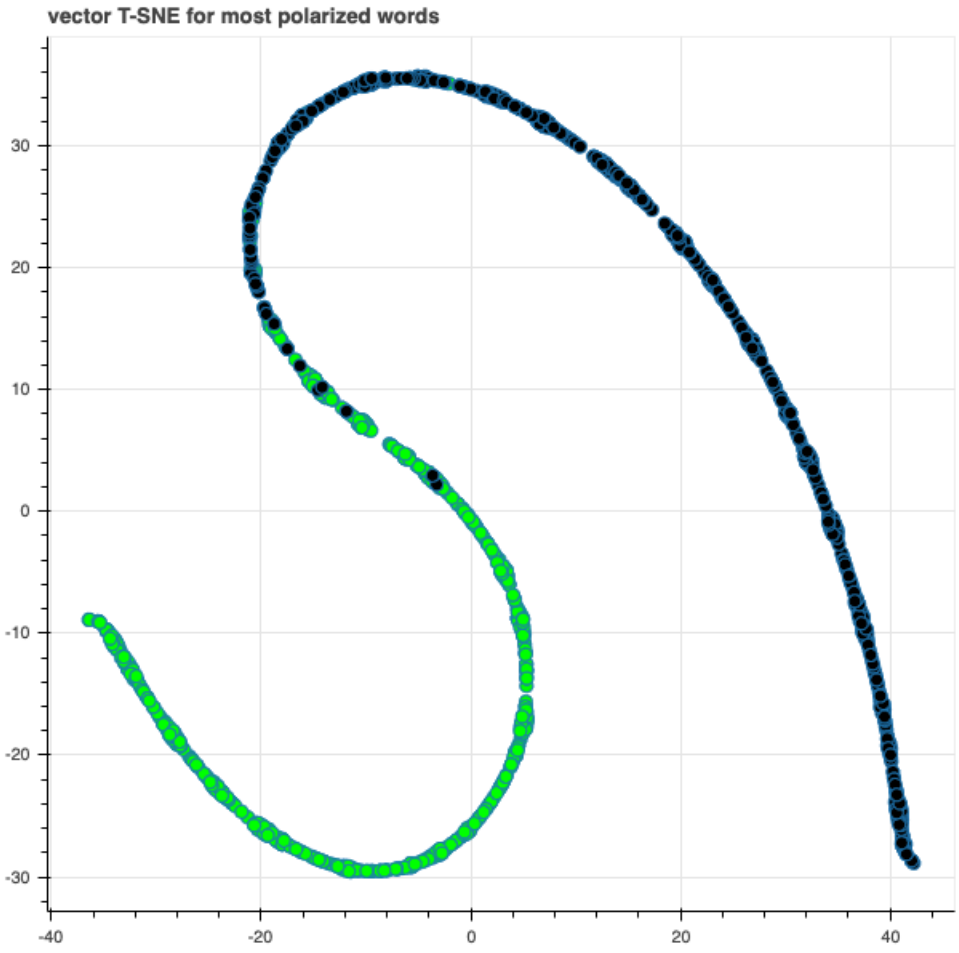
- Add the same two parameters (min_count and polarity_cutoff) and use them when we call pre_process_data

Training Accuracy = 85.6%, Testing Accuracy = 85.9%

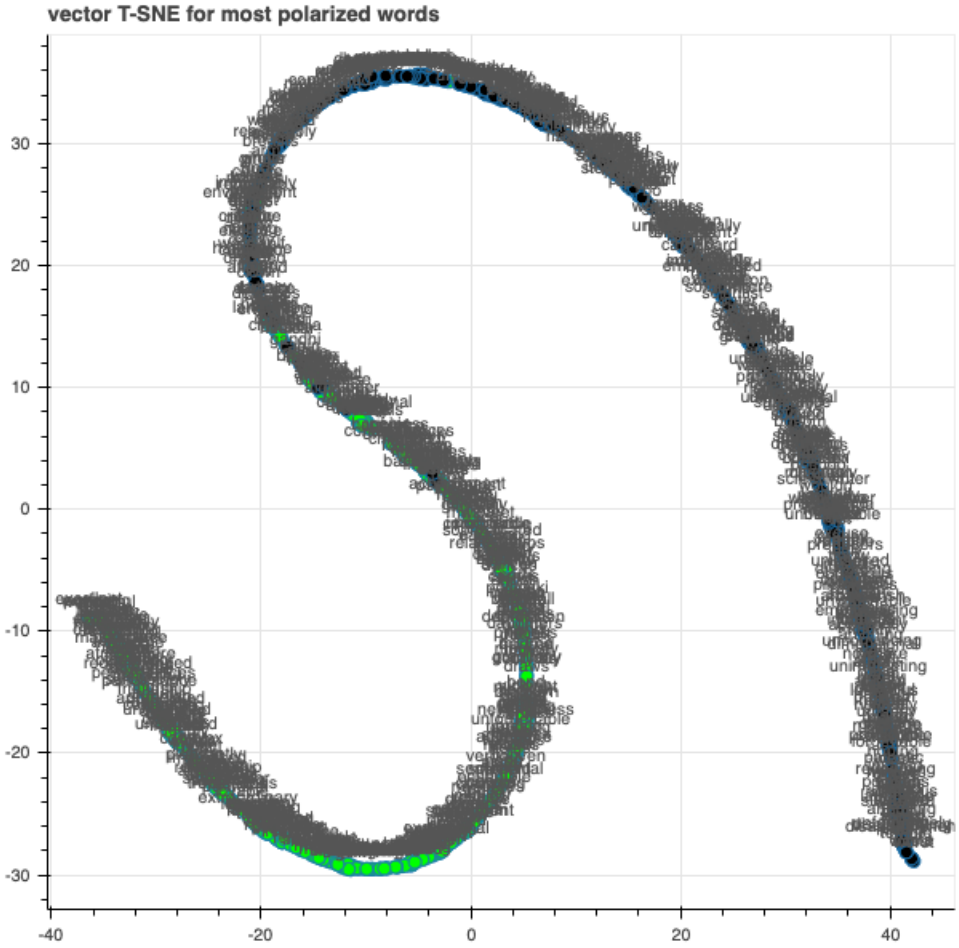
Visualization of Word Vectors as Positive or Negative

green indicates positive affecting words, black indicates negative affecting words

Graph-3



Graph-4



Visualization by adding layout with word-label that is which word is green in graph contributes to POSITIVE label and which word is black in graph contributes to NEGATIVE label.

Application

This model can be used to analyze whether a given product or anything in the market like a movie, any new startups, etc. has more positive influence or negative influence.

THANK YOU!