Lab 1 Report: Classifying Yelp Review Sentiment

1 Introduction

This lab focuses on training a review sentiment classifier. The dataset consists of a collection of comments, each labeled as either "positive" or "negative" to indicate the sentiment of the review. The dataset is divided into three subsets: training, validation, and test sets. We plan to use a Multi-Layer Perceptron (MLP) for classification, with features represented using the Bag-of-Words (BoW) approach. After training, we evaluate the model's performance using simple accuracy.

2 Task 1

2.1 *Using a term frequency*

The modification in Task 1.1 focuses on data preparation. The original code represents words in a review using a binary encoding (0-1), indicating the presence of individual words. It is important to note that the words included in our constructed dictionary (used for training) have already been filtered based on their frequency in the dataset. Specifically, only words that appear more frequently than a predefined threshold, `frequency_cutoff`, are retained. These "surviving" words are then represented as a vector, meaning that `ReviewVectorizer.vectorize()` returns a binary vector such as:

$$[0, 0, 1, 0, 1, \ldots, 0, 0]$$

Additionally, a mapping is maintained to preserve the relationship between integers and words.

When we apply a term presence, for one review, such as "this is a pretty awesome book, I have awesome reading experience!", we constructed a review vector with length of dictionary.

Review	**good**	**this**	**...**	**awesome**
review1	0	1	...	1
...	..	...	...	...

But if we use term frequency instead, the element in review vector will indicate the frequency of the current in the given review instead of its existence simply. With the same example, the vector will change to:

Review	**good**	**this**	**...**	**awesome**
review1	0	1	...	2
...	..	...	...	...

The accuracy of the modified version 1 model is 85.03, which is identical to the original model. I assume that since review texts tend to be relatively short, the frequency of each word in an individual review is often 1. As a result, considering our dataset is relatively small as well, the modified representation does not significantly alter the feature distribution, leading to similar classification performance.

2.2 *Adding additional hidden layer*

According to the original code, we can easily deduce that the model is as follows.

$$\hat{y} = \sigma(W_2 * \text{ReLU}(W_1 * x + b_1) + b_2) \tag{1}$$

Then, we apply Gradient Descent to find $W_1^*, b_1^*, W_2^*, b_2^*$ that minimize the loss function. In the original code, Binary Cross-Entropy Loss (`BCELoss`) is used.

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i log\hat{y} + (1 - y_i)log(1 - \hat{y})] \tag{2}$$

After added additional hidden layer, the model changed to:

$$\hat{y} = \sigma(W_3 * \text{ReLU}(W_2 * \text{ReLU}(W_1 * x + b_1) + b_2) + b_3) \tag{3}$$

Minimizing the loss function to get optimal $W_1^*, b_1^*, W_2^*, b_2^*, W_3^*, b_3^*$.

However, we get poorer model performance with accuracy at 84.64 compared to original 85.03. This decline could be attributed to several factors:

1. Increased Model Complexity: Adding an extra hidden layer introduces more parameters, which may require additional training data to generalize well. Given the limited dataset, the model might not have learned better representations.

2. Overfitting Risk: While deeper models generally improve learning capacity, they also increase the risk of overfitting, especially if the dataset is not sufficiently large. The additional hidden layer might capture noise rather than meaningful patterns.

2.3 *Changing frequency_cutoff*

As we analyzed in Section 2.1, the frequency filter becomes ineffective once we set frequency_cutoff to 0, resulting in a larger dictionary. The accuracy of this version reached 87.50, which is higher than all previous versions. Although the vector representation for each review became sparser—leading to increased computational cost—we retained more information. This likely explains the improved accuracy.

3 Task 2

The main goal of this task is to modify the model's output, transforming it into a multi-class classifier. The original model produced a single logit value, which was then passed through the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$ to map to $[0, 1]$ range, indicating the two categories: "negative" and "positive." After modification, the model outputs multiple scores (two in this case), corresponding to the number of categories. The predicted category is determined by selecting the one with the highest score.

First, we have to update our model, in the code is the class "ReviewClassifier". The original version is
nn.Linear(in_features=hidden_dim, out_features=1)
return torch.sigmoid(y_out).squeeze()

Now
nn.Linear(in_features=hidden_dim, out_features=2)
return y_out

Second, the training logic also need to change. The original loss function *BCELoss()* was replaced with *CrossEntropyLoss*. *CrossEntropyLoss* is able to compute multi-value loss. Instead of (2), it became

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c} log \hat{p_{i,c}} \tag{4}$$

While, C is the number of classification (2 here). $y_{i,c}$ is the label vector. And $\hat{p_{i,c}}$ is the probability after softmax normalization.

$$\hat{p_{i,c}} = \frac{e^{z_c}}{\sum_{j=1}^{C} e^{z_j}} \tag{5}$$

z_c here is the output logit value of the model.

Finally, we have to modify the accuracy computing way, typically focus on *y_pred*, such as
y_pred_list.extend((y_pred0.5).cpu().long().numpy())
change to
y_pred_list.extend(y_pred.max(dim=1).indices.cpu().numpy())