Bag of N-Gram Document Classification (DS-GA-1011)

Ziv Schwartz[1]

[1]New York University

October 10, 2018

I. Abstract

GitHub Repository: https://github.com/zivschwartz/Bag-of-N-Gram-Document-Classification

This assignment deals with the large movie review dataset from IMDB for binary sentiment classification. Within this large movie review dataset, there are 25,000 reviews designated for training and 25,000 designated for testing. Both of the training and testing sets are split up into an equal number of positive and negative reviews (12,500 within each sentiment directory). The master code follows the same layout as the report, with matching order and section headers. The master code can be found at my GitHub repository.

II. Data Loading & Preprocessing

After downloading the data into memory, the next step is to load the data and get it ready for classification. The first step is to define a function that opens and reads a text file from a given path. Using this function, another function is built that iterates through the train and test directories within the IMDB dataset. It extracts all of the relevant movie reviews and assigns a target parameter to each review based on sentiment: 1 if positive, 0 if negative. The train and test reviews, along with their target parameters, are extracted and are passed through a shuffler to ensure random sampling. The train dataset is split into an 80:20 training validation split (20,000 training reviews and 5,000 validation reviews). The testing dataset contains 25,000 reviews.

III. Baseline Model

My baseline model follows closely the example that was demonstrated in lab 3, no additional preprocessing or tokenization was done to establish the baseline model for comparison with different tokenization schemes and hyperparameter tuning.

A. Tokenization

In the baseline model, the packages spaCy and string were used to make the text lowercase and remove punctuation. Using this, a function was built to create tokenizations of the train, validation, and test datasets.

B. Vocabulary, PyTorch Loader, Bag of N-Gram Model

Following the tokenization, a vocabulary was constructed with a *default* size set to 10,000 that created a token and index scheme for the train, validation, and test datasets. Next an *IMDBDataset* class was built that takes in the indices and targets for each of the datasets and runs it on a PyTorch loader that returns an array of tensors: data list, length list, and label list. Using the token ID's and the PyTorch loader, a model is fit using Bag of N-Gram. This model takes in a *default* embedding size of 100.

C. Training the Bag of N-Gram Model

Now that the Bag of N-Gram model is fitted, we train the model using a *default* learning rate set to 0.01, 10 epochs, a Cross Entropy Loss criterion, and a *default* optimizer: Adam.

D. Model Results

Validation Accuracy: 83.48

Test Accuracy 80.448

IV. Model Optimization: Ablation Study

Having built and trained the *default* Bag of N-Gram model, we can now run an ablation study to optimize our model, improving validation and ultimately resulting in a better test accuracy. We look to alter Tokenization Schemes, Model Hyperparameters, and Optimization Hyperparameters. All parameters that are not being tested on are at their *default* values (Tokenization: spaCy Tokenizer with no preprocessing, n = 1, Vocab Size = 10,000, Embedding Size = 100, Optimizer = Adam, Learning Rate = 0.01, Linear Annealing).

A. Tokenization Schemes

Default: spaCy Tokenizer with no text preprocessing

- Test: Preprocess the movie review dataset by using regular expressions to remove HTML tags and any other non-text expressions. Pass the cleaned data to the spaCy tokenizer.
- Best tested Tokenization Scheme is the Default Scheme. **Validation Accuracy: 83.4**. The default tokenization scheme with no text preprocessing far outperformed the scheme with the cleaned and preprocessed data.

B. Model Hyperparameters

1. N-Gram n. (Default: n=1)

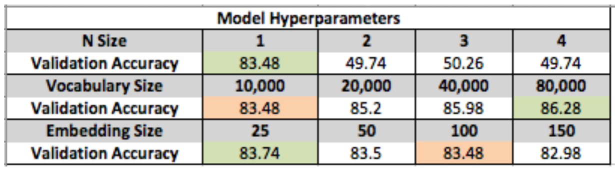
   - N value varied between 2, 3, 4
   - Best tested n size is 1 (Default). **Validation Accuracy: 83.4**. Choosing n = 1 is the simplest way to create tokens from text and was able to perform at a higher accuracy. N-grams allow for efficiency in scalability and can map to more common word phrases besides just simple unigrams. However, for this model, the unigram performed the best.

2. Vocabulary Size. (Default: Size 10,000)

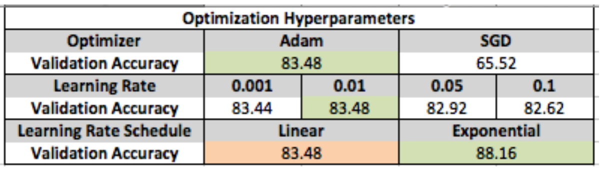
   - Vocabulary Size tested at [20000, 40000, 80000]
   - Best tested Vocabulary Size is 80,000. **Validation Accuracy: 86.28**. The validation accuracy was increasing for each iteration of increasing vocabulary size and at the same time so was the computation load. It is important to find a tradeoff. One possible correlator to vocabulary size could be the size of the dataset.

3. Embedding Size (Default: Size 100)

   - Embedding Size tested at [25, 50, 150]
   - Best tested Embedding Size is 25. **Validation Accuracy: 83.74.** Decreasing embedding size would produce better validation accuracy. Embedding size represents the data at a lower-dimension that can contain fewer cells and increase the model's capability.



Model Hyperparameters				
N Size	1	2	3	4
Validation Accuracy	83.48	49.74	50.26	49.74
Vocabulary Size	10,000	20,000	40,000	80,000
Validation Accuracy	83.48	85.2	85.98	86.28
Embedding Size	25	50	100	150
Validation Accuracy	83.74	83.5	83.48	82.98



Optimization Hyperparameters				
Optimizer	Adam		SGD	
Validation Accuracy	83.48		65.52	
Learning Rate	0.001	0.01	0.05	0.1
Validation Accuracy	83.44	83.48	82.92	82.62
Learning Rate Schedule	Linear		Exponential	
Validation Accuracy	83.48		88.16	

Figure 1: Tables of the Model and Optimization Hyperparameters and their respective validation accuracy. Green: highest validation accuracy. Orange: default validation accuracy (if lower than optimized).

C. Optimization Hyperparameters

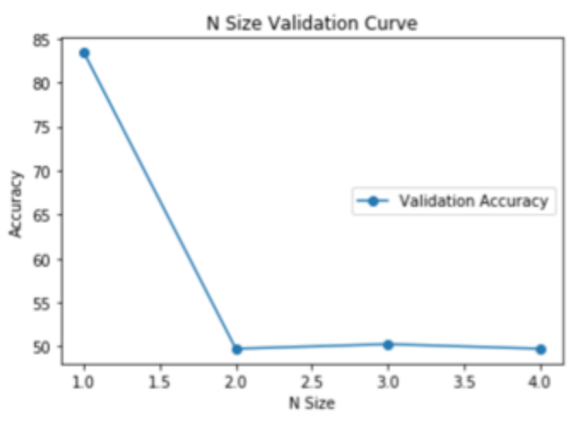
1. Optimizer (Default: Adam)
   - Test SGD Optimizer
   - Best tested Optimizer is Adam (Default). **Validation Accuracy: 83.48.** The Adam Optimizer far outperforms the SGD Optimizer in increasing validation accuracy. Some benefits of Adam include that it is straightforward and computationally efficient. It differs from classical stochastic gradient descent because it can handle sparse gradients on a multitude of problems by utilizing a per-parameter learning rate.
2. Learning Rate (Default: LR = 0.01)
   - Learning Rate test at [0.001, 0.05, 0.1]
   - Best tested Learning Rate is 0.01 (Default). **Validation Accuracy: 83.48.** Learning rate is a hyper-parameter that deals with weight adjustments in regards to the loss gradient. As the value decreases, the slower the movement along the downward slope of the loss gradient. For this model, the learning rate of 0.01 and 0.001 are very close but 0.01 is slightly higher and is more efficient on computation.
3. Learning Rate Schedule (Default: Linear Annealing)
   - Test Exponential Decay Learning Rate Schedule
   - Best tested Learning Rate Schedule is Exponential Decay. **Validation Accuracy: 88.16.** While there are several different types of learning rate schedules, I chose to compare the linear annealing with an exponential decay and the validation accuracy was significantly higher. The exponential decay learning rate schedule passes a gamma parameter (multiplicative factor of exponential decay) which I set to 0.1, a common value for gamma.

V. Optimized Model

After running an ablation study we run a new model with the optimized hyperparameters.

Best performing hyperparameter values:

- Tokenization: spaCy Tokenizer with no preprocessing, n = 1, Vocab Size = 80,000, Embedding Size = 25, Optimizer = Adam, Learning Rate = 0.01, Learning Rate Schedule = Exponential Decay.
- The Optimized Model results in a **Test Accuracy of 85.868.** Compared to the default model (test accuracy = 80.448), test accuracy rose by more than 5 points, ensuring that this model performs to a higher degree of accuracy.



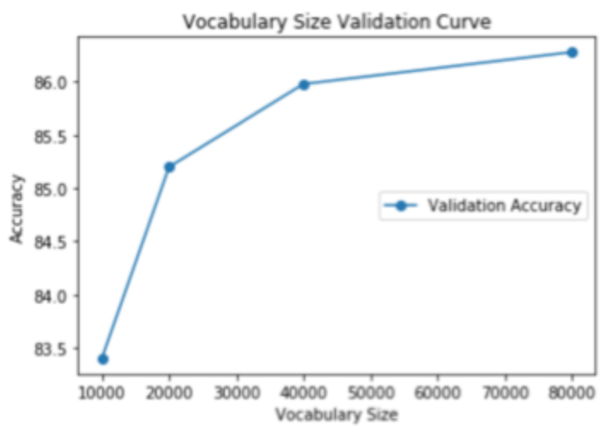
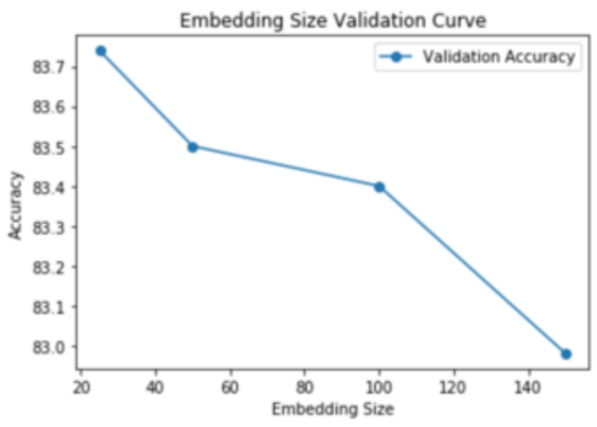


Figure 2: Validation Curves for N-Gram and Vocabulary Size



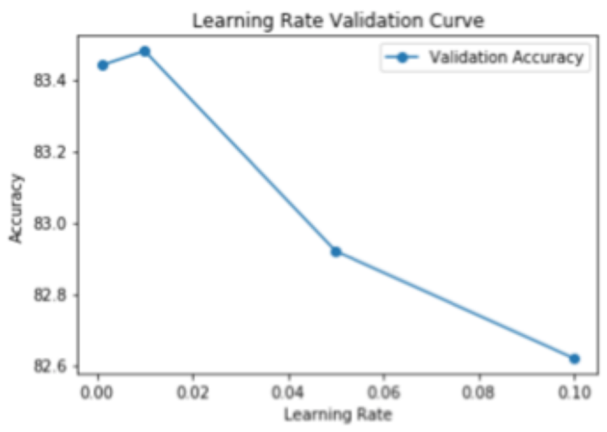


Figure 3: Validation Curves for Embedding Size and Learning Rate

VI. Predictions from the Optimized Model

Using this model we can also determine the correctness of the predictions on the validation dataset. Comparing the first 6 values of the predicted values to the validation target datasets, we observe the following:

- predictions = (1,0,0,1,0,1)
- val_target[:6] = (1,1,1,1,0,0)

Matching the values between the two determines the level of correctness.

- Correct predictions: 1st, 4th, and 5th review
- Incorrect predictions: 2nd, 3rd, and 6th review

The actual reviews and their polarity can be found in the Master Code file in the GitHub repository.