DS-GA 1011, Fall 2017

Homework 1: Bag of N-Grams Document Classification and Hyper-parameter Exploration

September 28, 2017

1 Introduction

In this assignment, we will tackle the problem of sentiment analysis using an n-gram classifier. Throughout this assignment, you will learn:

- how to extract n-gram features from raw text
- how to perform text classification using n-grams
- how to analyze and tune hyper-parameters in a model

In the task of **Document Classification**, we are given a document d from a document space $\mathbb{D}$ and try to predict the class c within a class space $\mathbb{C} = \{c_1, c_2, ..., c_k\}$ that d belongs to. In this assignment, we will focus on Sentiment Analysis, in which we are asked to identify the attitude in a document. The class space $\mathbb{C} = \{positive, negative\}$ and document space $\mathbb{D}$ can be text data such as reviews, emails and speech. More specifically, we will be using the Large Movie Review Dataset [3] on IMDB movie reviews. You can download the dataset from this link. This dataset is composed of 25,000 training samples and 25,000 test samples. Each sample is labeled with either *positive* or *negative*. In Part I of the template, we provided code that loads the dataset from the disk and divides the 25,000 training data into a training set of 23,000 and validation set of 2,000. Note that the label distribution is intentionally kept balanced in the data loading code.

2 Bag of N-grams (20 pts)

The first step to deal with text data is to transform them into logic structure that computers can "understand". For the problem of Sentiment Analysis, we will use the trick of Bag of N-grams. A (word-level) N-gram can be thought of as a continuous sequence of tokens in a document. For instance, given a document "I love NLP", if we tokenize it using space, we will have unigrams $\{I, love, NLP\}$, bigrams $\{(I, love), (love, NLP)\}$, and trigram $\{(I \quad love \quad nlp)\}$.

To represent N-gram, we will first set a vocabulary base V which can be viewed as a set of words. Since we are limited by the available computational resources, we need to set a limit on the number of vocabularies we add to our vocabulary base. A common way to construct vocabulary base is to choose the top k (a hyper-parameter set by user) most frequently occurred N-grams in the data set.

Once we build up our vocabulary base, we need to use word embedding vectors to represent each N-gram in V. In a word, a word embedding matrix $E \in \mathbb{R}^{|V| \times d}$ maps each word i in V to a vector $E_i \in \mathbb{R}^d$. Every time we meet word i in the data, we feed E_i into our model. For more information about word embedding, please refer to Chapter 10 and 11 in the Goldberg text book [1].

In Part II of our template, we provide you with a top-level function *process_text_dataset* that transforms a collection of text data points into a collection of n-gram indices so that our model knows which embedding E_i to use for each N-gram in the input document. Your job is to fill in the code for the help functions that complete this process.

2.1 Extract N-gram from Text (10 pts)

Function $extract\_ngram\_from\_text$ takes two inputs: a raw text string *text* and an integer n which represents the max length of continuous tokens we would like to extract from *text*. For input 'I love NLP' and $n = 2$, this function should extract all unigram and bigrams $tokens = \{I, love, NLP, (I, love), (love, NLP)\}$. Note that in addition to *tokens*, this program should also output an Python Counter object that counts the number of occurrences for each N-gram in *tokens*. Please fill in your code to complete this function.

2.2 Construct Vocabulary Base (5 pts)

Once we obtain the N-grams for each sample, we can construct our vocabulary base. Function *construct_ngram_indexer* takes in two arguments *ngram_counter_list*, which is a collection of Python Counter mentioned in the previous problem, and an integer *topk*. This function should form a vocabulary base using the most common *topk* N-grams. Moreover, we would like to take a further step and create a dictionary *ngram_indexer* that maps each N-gram in our vocabulary base to a unique integer that represents the N-gram's identity. Note that index 0 is reserved for special padding symbol which will be explained later. Please fill in your code to complete this function.

2.3 Map N-gram to Index (5 pts)

Lastly, function *token_to_index* takes in *tokens*, which is a list of N-grams, and *ngram_indexer* constructed above and maps each N-gram in the list to its corresponding index. Please fill in your code to complete this function.

3 Fast Text Implementation (20 pts)

In this section, you will implement the well-known FastText model [2] using PyTorch. Please read this paper (very short) and make sure you understand the logic before coding. In Part III, we provide you with the code that loads the N-gram dataset into the PyTorch model. Since the length of each document varies, we pad each document vector in each mini-batch to ensure that they have the same length. The technique of padding usually refers to adding a special token PAD to fill in the sentences so that all samples in the mini-batch will have the same length. In our model, we assign index 0 to the special PAD token. In addition, please see PyTorch's documentation on nn model (especially torch.nn.Embedding function).

In Part IV - VI, we provide you with necessary template to define, train, and test your model. The original paper reports an accuracy of 88% on the IMDB dataset. Since we use less data and lack some of the optimization techniques mentioned in the paper, our model might achieve a slightly lower test accuracy. **In order to obtain full score for this part, we expect your implementation to achieve at least** 85% **of accuracy on the test set using the default hyper-parameters**. Please fill in your code to complete the implementation.

4 Model Analytics (20 pts)

4.1 Evaluation Metrics (5 pts)

In this assignment, we used Binary Cross Entropy (BCE) loss in training while using accuracy as the reporting metrics. Why don't we directly use accuracy in the loss function for training?

$$BCE = -\frac{1}{n}\sum_{i=1}^{n} y_i log(\hat{y}_i) + (1 - y_i)log(1 - \hat{y}_i) \tag{1}$$

Moreover, accuracy, as a reporting metrics, would suffer from a serious drawback if the validation / test data set satisfies certain property. Please point out the potential drawback of accuracy and why we don't need to worry about it in our IMDB dataset?

4.2 Error Analysis (15 pts)

Please list 3 false positive examples and 3 false negative examples. Could you qualitatively state the patterns of the errors that your model makes. What are the reasons that your model makes mistakes on these samples? (eg. the model can't recognize that quotation marks sometimes flip the word's meaning and it's not able to do this because punctuation is removed in the feature engineering stage.)

5 Hyper-parameters (40 pts)

5.1 Analyze Hyper-parameters (15 pts)

Please qualitatively state how would *learning_rate*, *embedding_dim*, and *vocabulary_size* affect the model performance and why. Furthermore, please include plots to empirically verify your statements on each of these 3 hyper-parameters (eg. learning curves for different learning rates).

5.2 Tune Hyper-parameters (15 pts)

Try each combination of $(embedding\_dim, vocabulary\_size)$ where $embedding\_dim \in \{10, 20, 50, 100\}$ and $vocabulary\_size \in \{500, 5000, 10000, 20000, 30000\}$. Plot the training accuracy, validation accuracy, and test accuracy. What's the optimal set of hyper-parameters? You can plot one of the hyper-parameter on x-axis and represent values of the other hyper-parameters as lines of different colors. The y-axis should be accuracy.

5.3 Early Stop (10 pts)

One way to implement the idea of early stop is to terminate the training process if the model's validation performance is not increased by at least ϵ for at least t training steps. Implement early stop using the template code provided in Part VI. Please also briefly explain why early stop can serve as a regularization method.

6 Logistics

Please turn in both your Python notebook file and a pdf report (preferably in Latex) with answer for all the questions above on NYU Classes. **The due date is 6:45 pm on 10/04/2017.**

References

[1] Goldberg, Y. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies 10*, 1 (2017), 1–309.

[2] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).

[3] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (Portland, Oregon, USA, June 2011), Association for Computational Linguistics, pp. 142–150.