Natural Language Processing, Assignment 3

October 18th, 2022

1 Introduction

In this assignment, we will work on a real world dataset Yelp for classification. The ultimate goal of your job is to predict the customer rating for businesses like restaurants based on the customer review text. Hopefully, after this homework you can know the general approach to solve a text related machine learning problem and understand what kind of challenges you may meet when working on a real world dataset.

You are required to finish this homework in Python 3. Use of deep learning frameworks (PyTorch, TensorFlow, Keras, etc) is allowed. It is fine to use APIs from these frameworks, but you need to build the model by yourself, i.e., do not copy any code or download any model that has been released. **You should submit the report, your code, a readme file, and the scripts that can reproduce all your experiments at the same time in a .zip file!**

2 Train a Classifier Model

In this section, we will introduce the dataset and the basic classification model.

2.1 Dataset

We provide a subset of the Yelp dataset, with 6000 training examples, 500 valid examples, and 500 test examples. The dataset can be downloaded from https://cloud.tsinghua.edu.cn/f/1031674e94be47529bbe/ and includes train.csv, valid.csv, test.csv, and vocab.txt. The training, validation, and testing files all contain a list of paragraphs and their label. The labels are from 1 to 5, representing the customer rating for a business. You should use the "train.csv" for training, "valid.csv" for validation (model selection and hyperparameter tuning), and "test.csv" for testing. The vocabulary file contains an array of strings. Each string is a word in the vocabulary.

We have provided scripts for data preprocessing.

2.2 Classification Model

Here we only introduce a simple classification model, which can then be extended using sequential models. Let $c \in \{1, 2, ..., C\}$ be the class-label indicator. The problem of text classification is to estimate the probability of the class as:

$$P(y = c|x, W) = \frac{e^{w_c^T x}}{\sum_{c'=1}^{C} e^{w_{c'}^T x}}$$

where x is the feature vector of a sentence, $y \in \{1, 2, ..., C\}$ is a class label, and W is the model parameter matrix whose columns are w_c for $c \in \{1, 2, ..., C\}$. Given a training set of labeled pairs $D = \{(x_i, y_i)\}_{i=1}^{n}$, we maximimize the log likelihood of data:

$$l(W) = \sum_{i=1}^{n} logP(y_i|x_i, W)$$

It is possible to add a regularization term (for example, L2 regularization) to prevent overfitting. You might also try dropout or different optimization schemes as we mentioned in Assignment 2.

2.3 Evaluation

As we have done in Assigiment 2, you should plot the curve of the training loss and validation loss during the training process. After picking the best model checkpoint via the validation set, you should report its performance (Accuracy) on the validation set and test set.

3 Problems

Each of the problems below describes a classification method. The following contents should be reported.

1. For each problem, you should describe the details which you think are important in your experiment, for example, the hyper-parameters you choose.
2. For each problem, plot the learning curve of the training set and validation set. Report the performance on the validation set and the test set.
3. Compare the results of the these methods with analysis. Discuss their pros and cons.

We have provided a code framework, and you only need to fill in the missing parts in each file to complete this assignment.

Problem 1 : (25 points) **Bag of Words (BOW)**

As a baseline approach, let us use a bag of words to represent each document, i.e., using a bag of tokens and the corresponding counts in each review and ignoring the order of tokens. For example, if the vocabulary is [a, b, c, d, e, f, g], and a training instance contains tokens [a:2, b:2, d:1, g:1], your feature vector will be [2, 2, 0, 1, 0, 0, 1]. Note that the order of tokens in the vocabulary does not matter, but you should make sure the order is the same for all the instances.

Note that, in this setting, to achieve better performance, you should do the following preprocessing.

1. Remove the stop words first. The stop words could be found in NLTK (*"from nltk.corpus import stopwords"*).
2. Remove all the punctuation in the tokens (e.g., "it's" will become "its", "?" or "!" will become "").
3. Remove all the tokens that contain numbers.

You should train with the following two steps:

1. Construct the bag-of-words features.
2. Feed the constructed features into the linear classification model described in Section 2, and then report the results.

Problem 2 : (25 points) **FastText**

Implement the FastText classifier we introduced in the lecture. The training steps are as follows:

1. Use bag of n-grams as additional features to capture information about the local word order.
2. Use hashing[1]. For example, let the size of the vocabulary be 10000 in this model. Using a hashing function, we can map the 10000 tokens into a vector with a much smaller dimension (i.e., 1000

[1]You can refer to the paper "Enriching Word Vectors with Subword Information" for more details.

dimensions in your implementation).

3. Convert word and n-gram features into embeddings. Average the embeddings of features to form the hidden states.
4. Feed the hidden states to a fully-connected softmax layer for multi-class classification.

Problem 3 : (25 points) **CNN**

Implement a TextCNN classifier[2] we introduced in the class. The training steps are as follows.

1. Convert each word in the sentence into word vector $x_1, ..., x_n$ (with random initialization), where x_i is a k-dimensional word vector.
2. Construct features via CNN. A 1-d convolution operation involves a filter $w \in \mathbb{R}^{hk}$, which is applied to a window of h words to produce a new feature. For position i, $c_i = f(w^T x_{i:i+h-1} + b)$, where f is the nonlinear function and b is the bias term. Concatenate all possible features generated by different windows and then apply the max-pooling to get one feature. Use multiple filters (with varying window sizes) to obtain multiple features.
3. Feed these features to a fully-connected softmax layer.

Problem 4 : (25 points) **Bi-LSTM**

Implement a classifier with a bidirection LSTM. This method is similar to that of TextCNN. The only difference is how the features are produced (via convolution or LSTMs).

The training steps are as follows.

1. Convert each word in the sentence into word vector $x_1, ..., x_n$ (with random initialization), where x_i is a k-dimensional word vector.
2. Construct features via a bidirectional LSTM. Feed the inputs $(x_1, ..., x_n)$ into the bidirectional LSTM layer to obtain features. You can experiment with the techniques introduced in the class. For example, you could experiment with stacking several layers, conducting mean/sum/max pooling, or trying using the first or last state.
3. Feed these features to a fully-connected softmax layer.

[2]You can refer to the paper "Convolutional Neural Networks for Sentence Classification" for more details.