Udacity Machine Learning Engineer

YongKyung Oh

Project Report

Abstract

Sentiment classification, detecting if a piece of text is positive or negative, is a common NLP task that is useful for understanding feedback in product reviews, user's opinions, etc. Sentiment can be expressed in natural language in both trivial and non-trivial ways.

In this project, I build two sentiment classifiers based on Naive Bayes and neural networks. Because of the lack of train data, I conducted pre-process and random sampling. After that, I develop two classifiers. For Naive Bayes, I use the TF-IDF as feature of train data. The main model is based on MultinomialNB. In case of neural net, I compared word embedding from train data and pretrained embedding(Glove 6B). The main model is based on bi-directional LSTM. Both classifiers work well and acheive the target accuracy.

Datsets and Inputs: dev_text.txt contains reviews, one per line, and dev_label.txt contains their labels. I use these files for training and testing models, by splitting them. heldout_text.txt contains reviews that the models will be evaluated on the unknown label data. In this project, I will just show the performance of model on the dev data.

Solution Statement: I will build a RNN model for the sentiment classification. Specifically, I will build model using pytorch and torchtext. Pretrained word embedding will be used.

Benchmark model: There are two different benchmark models. First, naïve Bayes model will be the baseline model for comparison. Second, Word-to-Vector model will be compared with pretrained embedding model (Glove6B).

Evaluation metrics: For simplicity, the accuracy on the text and label will be the metric for model comparison.

Project Design

1. Preprocess: Train data is consist with 1000 pos-labeled review and 1000 neg-labeled review. Through preprocess, I divided sentence into tokens and remove uninformed text using nltk. Also, I removed all numeric text. After preprocess, dataset only contain useful tokens. Average token per sentence is 216.31 where min is 15 and max is 1654.

2. Resampling: The number of train data is not big. So, I implemented permutation resampling to select more samples. Through this step, I randomly select the same number of samples with original dev data. So, the total number of train data is 4000 and the train/validation ratio is different in each methods.

3. Strategy for data usage: After data clean-up, I split the train data into train set and validation set. I selected 20% of data. So, 1600 for train set and 400 for validation set. This data is used to evaluate the model. After that, I conducted random sampling for train set. Train set is not enough for each class. So, I shuffle the each class set and select random samples as follow. Through this process, I can get 3600 train data with 1800 pos-labled samples and 1800 neg-labeled samples. I use the validation for 400 samples.

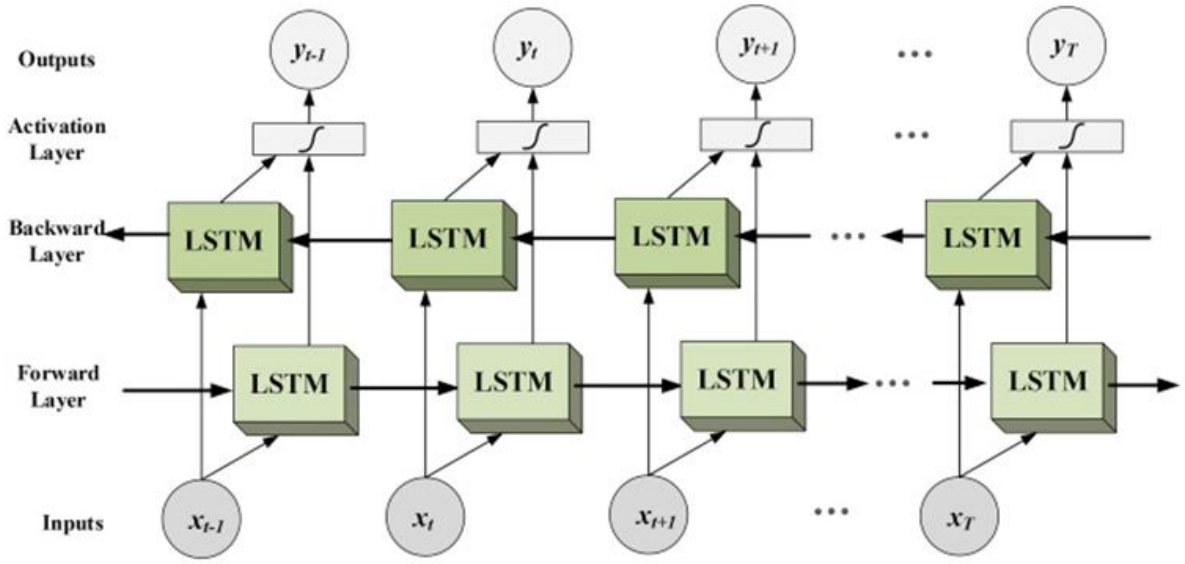
4-1. Naïve Bayes model: Most of all, I build a vocabulary dictionary from both classes. I selected top 2000 words from each classes and use the unique 2500 words as dictionary. Main model is multinomial NB which is suggested by J. Rennie et al. (2003. Also, I applied 10-fold validation to select the best model.

- Bag of Words (BoW) Model: BoW model creates a vocabulary extracting the unique words from the documents and keeps the vector with the term frequency of the particular word in the corresponding document.
- TF has the same explanation as in BoW model. IDF is the inverse of number of documents that a particular term appears or the inverse of document frequency by compensating the rarity problem in BoW model. BoW has some limitations. Term ordering is not considered and Rareness of a term is not considered. Therefore, TF-IDF is better option for me to build model.
- Most of all, train data (and validation data) don't have enough number of tokens. When I develop a model, the accuracy can be increased. But it means that the model is

overfitted. Also, this model can't evaluate unseen tokens. Therefore, I use the TF-IDF that can better perform in this situation.

	Train	**Validation**
NB	0.90375	0.80000
NB K-fold	0.91938	0.83000

4-2. RNN model: Main model is bi-directional LSTM, which is appropriate for the sequential data such as text. Model only has two hidden layer with 100 nodes and each layer represent forward and backward characteristics. Output node is 2 and I use the logit for the output value. Rather than other activation function, I used logit and cross entropy to evaluate loss.



RNN(

(embedding): Embedding(9310, 100)

(rnn): LSTM(100, 100, num_layers=2, batch_first=True, dropout=0.5, bidirectional=True)

(fc): Linear(in_features=200, out_features=2, bias=True)

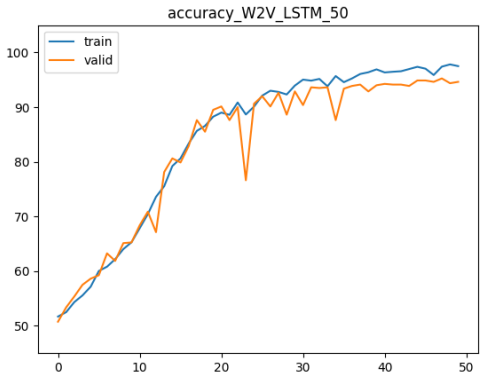
(dropout): Dropout(p=0.5, inplace=False)

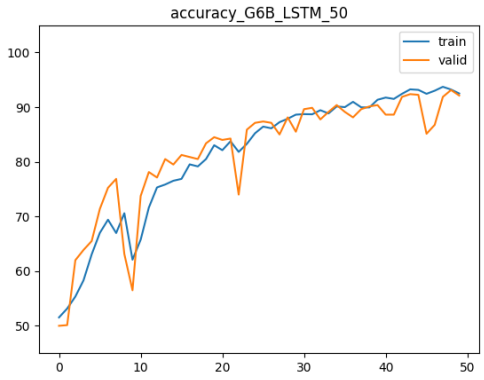
)

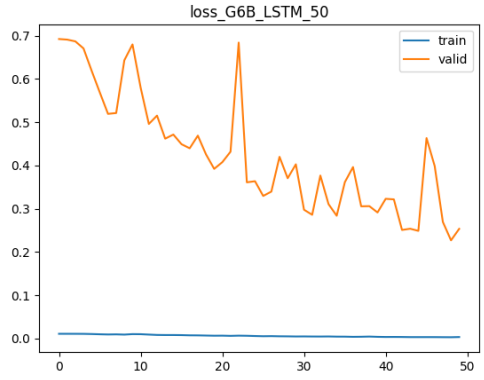
I use the word embedding as a feature for the model.In the data process, I set the number of words in the sentence as 1000. So, if the word is shorter than 1000, remain features are filled with padding. Each embedding has 100 dimensional feature through word2vec model. I use 2 different word embedding: word2vec from train data and pretrained embedding glove 6B. Pretrained embedding is work better, because train data is not sufficient for dealing with new inputs.

To avoid the overfitting, I added dropout layer in the model. Batch size is 64 and maximum epoch is 50. Train set has 50 batches (=3200/64) and validation set has 13 batches (=800/64). I used the adam optimizer with decaying factor: learning rate=1e-4 and decaying=1e-6.

	Train	**Validation**
NB	0.90375	0.80000
NB K-fold	0.91938	0.83000
LSTM W2V	0.95969	0.93125
LSTM Glove 6B	0.94188	0.92250







Here I summarize the accuracy and loss result of the suggested models. W2V tends to show slightly better than G6B, but I thought that this result is too overfitting on the data. Therefore I used the glove 6B model as final model. Also, I got the full score in the leaderboard, which means prediction accuracy for heldout data is higher than 0.7 in naive bayes and 0.87 in neural net.

Compare to the naive bayes, LSTM has higher score in the validation set because of the order between data split and resampling. Strictly, resampling should be conducted after data split. However it shows better results in LSTM, because the heldout data has really similar characteristics with dev data. Also this approach work as K-fold cross validation on the batch level. I tried this approach to handle the small data issue in neural net. So, the model can be overfitted to the data and may not work well with unseen data (or new inputs). Therefore, more train data will improve the suggested approach and generate better results.