IITP-03 Assignment 2: Sentiment Classification

YongKyung Oh[*]
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
`yongkyuo@andrew.cmu.edu`

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 assignment, 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.

1 Preprocess

1.1 Data Clean-up

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.

1.2 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.

> A resampling-based method of inference—permutation tests—is often used when distributional assumptions are questionable or unmet. Not only are these methods useful for obvious departures from parametric assumptions (e.g., normality) and small sample sizes, but they are also more robust than their parametric counterparts in the presences of outliers and missing data [1].

2 Task 1

Implement and train a Naive Bayes classifier on dev_text.txt and dev_label.txt. Run the trained model on heldout_text.txt and generate output file heldout_pred_nb.txt that contains predicted labels in the same format as dev_label.txt.

```
python naivebayes.py dev_text.txt dev_label.txt heldout_text.txt heldout_pred_nb.txt
```

[*]UNIST, ok19925@unist.ac.kr

2.1 How did you use the dev data?

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.

- random shuffle pos-labeled dataset and neg-labeled dataset.
- select random 100 samples in each dataset
- append to original dataset
- iterate 10 times

Dev data (2000) -> Split:Train(1600) / Valid(400) -> Resampling: Train(3600) / Valid(400)

2.2 How did you preprocess the data?

I explained it in the previous section. Most of the preprocess are tokenization and resampling. First part is essential for extracting features from the text. I use the nltk package to tokenize and clean-up the text data. Second part is increasing the train data to build better model. To avoid the overfitting, I use the validation set to evaluate it.

2.3 What is the Feature for model?

- What is the feature? Why do you think it makes sense to use this feature? [2]
  - 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.
- Did the feature improve accuracy, either on the heldout or on your own test set? Why do you think it did or did not improve accuracy?
  - 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.
  - Through preprocess and resampling, model can utilze more features from each class. At the same time, model can be overfitted to the train data (and validation data). However I assumed that heldout data is also similar charateristic with train data, so build a model with current strategy.

2.4 Details of Naive 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)[2]. Also, I applied 10-fold validation to select the best model.

> Naive Bayes classifier for multinomial models[3]: The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work. [3].

[2]BoW vs TF−IDF in Information Retrieval

[3]sklearn.naive_bayes.MultinomialNB

2.5 What is the final accuracy on your own test set?

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

Table 1: Naive Bayes Results

Oveall accuracy for train data is around 90% and validation is around 80%. Also, K-fold validation improve the performance slightly.

3 Task 2

Implement and train a neural classifier on dev_text.txt and dev_label.txt. Run the trained model on heldout_text.txt and generate output file heldout_pred_nn.txt that contains predicted labels in the same format as dev_label.txt. More specifically, we expect your source code file neuralnet.py to take the same four input arguments as in Task 1. We will not run this code on Gradescope, so you must submit heldout_pred_nn.txt.

```
python neuralnet.py dev_text.txt dev_label.txt heldout_text.txt heldout_pred_nn.txt
```

3.1 How did you use the dev data?

Compare to Task 1, I conducted resampling first. During the training, data is shuffled into batch iterator. Therefore I use the sampling first even the train set and validation set are slightly overlapped. I conducted random sampling for train data. Train data is not enough for each class. So, I shuffle the each class set and select random samples as follow. After that, I split the train data into train set and validation set. I selected 20% of data. Through this process, I can get 3200 train data with 1600 pos-labled samples and 1600 neg-labeled samples. I use the validation for 800 samples.

- random shuffle pos-labeled dataset and neg-labeled dataset.
- select random 100 samples in each dataset
- append to original dataset
- iterate 10 times

Dev data (2000) -> Resampling: Dev data (4000) -> Split:Train(3200) / Valid(800)

Strictly, this approach may cause problem, because the train set and validation set may be overlapped. I tried to avoid it in the batch level. It can be work as K fold cross validation in the training phase. By using this small data, I need to utilize all data into both training and validation. Therefore the performance on the validation set is higher than naive bayes. It may be overfitted into the data. So, I use 0.5 dropout to build a general model. At the same time, the test data would be similar characteristics with development data (train data and validation data), so this strategy works to get a higher accuracy.

3.2 What is the architecture of your model? What is the input feature of the model?

```
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)
)
```

> The basic idea of bidirectional recurrent neural nets (BRNNs) is to present each training sequence forwards and backwards to two separate recurrent nets, both of which are connected to the same output layer. This means that for every point in a given sequence, the BRNN has complete, sequential information about all points before and after it.

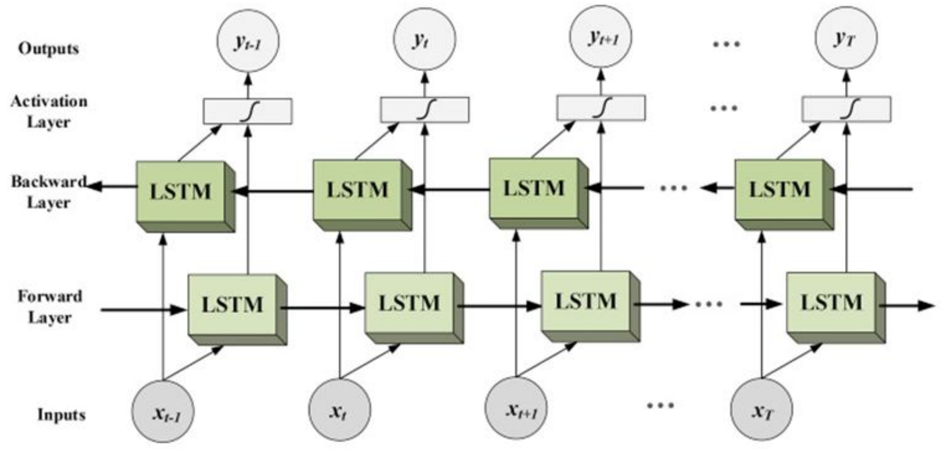

Figure 1: Structure of Bi-directional LSTM

> Graves et al.(2005) have found that bidirectional networks are significantly more effective than unidirectional ones, and that LSTM is much faster to train than standard RNNs and MLPs, and also slightly more accurate.[4]

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 fuction, I used logit and cross entropy to evaluate loss. Basic architecture is referred by the github reference [4]

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.

3.3 What is the final accuracy on your own test set?

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.

	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

Table 2: Results

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

[4]pytorch-sentiment-analysis

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.

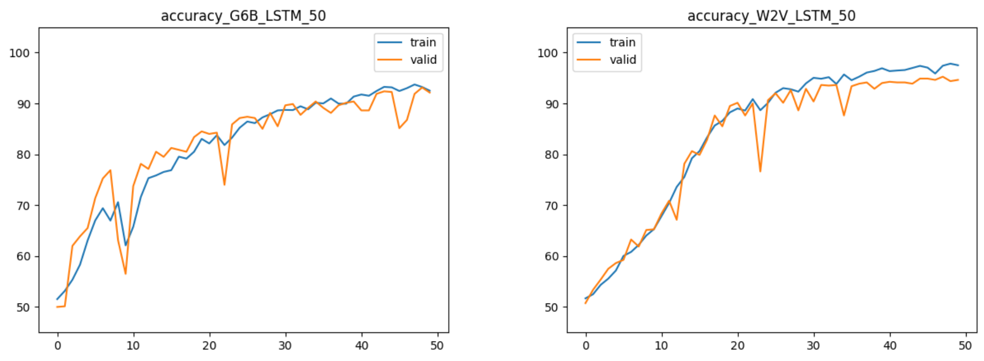

Figure 2: LSTM accuracy comparison: Glove 6B vs Word2Vec

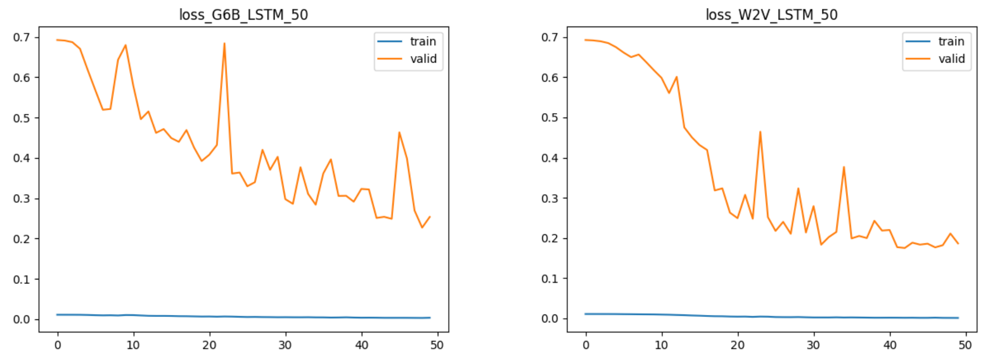

Figure 3: LSTM loss comparison: Glove 6B vs Word2Vec

References

[1] Bonnie J LaFleur and Robert A Greevy. Introduction to permutation and resampling based hypothesis tests. *Journal of Clinical Child & Adolescent Psychology*, 38(2):286–294, 2009.

[2] Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 616–623, 2003.

[3] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge university press, 2008.

[4] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.