Sentiment Analysis on Amazon Product Reviews by Using Long Short-Term Memory Neural Networks and fastText Classifiers

Yi Li, yl808@georgetown.edu

Abstract:
This paper uses labeled Amazon product reviews to predict the sentiment of a sequence of words. I used two methods to do the prediction: one is by building long short-term memory (LSTM) neural network models, and the other is by building a fastText classifier which is much more efficient in computation and much easier to build than the LSTM models. The best LSTM model I trained got a test accuracy rate of 0.88, and I used 160k reviews as the training dataset and 40k reviews as the test dataset for this model. The manual test results show that this model is good enough for real-world applications as the results are quite reasonable. For the fastText classifier, I used 3.6 million reviews as the training dataset and 400k reviews as the test dataset, and I got the test accuracy rate of 0.91. The manual test results show that this classifier gives the same sentiment predictions for the positive and the negative samples as the best LSTM model does, but it cannot identify the neutral samples as the best LSTM model does.

1. Introduction
This paper uses the labeled Amazon product reviews from Kaggle [1] to predict the sentiment of a sequence of words. I used two methods to do the prediction: one is by building long short-term memory (LSTM) neural network models, and the other is by building a fastText classifier developed by Facebook AI Research [3,4]. The paper [3] says "The fastText classifier is a simple and efficient approach for text classification, and is often on par with deep learning classifiers in terms of accuracy, and many orders of magnitude faster for training and evaluation." Both the LSTM model and the fastText classifier take a sequence of words as an input. The LSTM model outputs a value between 0 and 1 to indicate the score of the sentiment while the fastText produces a probability distribution over the predefined classes, which are "positive" or "negative" in this paper.

2. Data
2.1 Amazon product reviews
This paper uses the labeled Amazon product reviews from Kaggle [1] to train and test the LSTM models and the fastText classifiers. There are two datasets: the dataset "train.ft.txt" contains 3.6 million Amazon text reviews and ratings (1.6GB); the dataset "test.ft.txt" contains 400k Amazon text reviews and ratings (177.4MB). The ratings here can be either positive (represented as "__label__2" in the file) or negative (represented as "__label__1" in the file). The following is one sample from the file "test.ft.txt":

"__label__1 Batteries died within a year ...: I bought this charger in Jul 2003 and it worked OK for a while. The design is nice and convenient. However, after about a year, the batteries would not hold a charge. Might as well just get alkaline disposables, or look elsewhere for a charger that comes with batteries that have better staying power."

2.2 Pre-trained word vectors
The pre-trained word vectors come from Google News dataset that contains over 100 billion different words, each with a dimensionality of 300 [2]. However, this original file GoogleNews-vectors-negative300.bin.gz (1.5GB) is too large for my laptop. So instead, I only used the first 100k words, each with a dimensionality of 300. For the LSTM models, I need to use these pre-trained word vectors as the weights of each word and input or embed these weights into the LSTM models, so that the models can be trained. In case there are words or typos in the text reviews that they cannot be matched from the pre-trained dataset, I added an index for unknown words to the dataset. I also added an index of Null to the dataset, so

that if one sentence is too short, I can add Nulls to the end of the sentence to keep each input sentence the same length. For fastText classifiers, they do not need pre-trained word vectors as they would automatically generate their own word vectors.

3. Methods

3.1 LSTM models

I used the Python package Keras [5] with TensorFlow as the backend to build long short-term memory (LSTM) neural network models. I tried loading different numbers of records and always used the first 80% of the records for training and the rest 20% of the records for testing. As this is a supervised learning model, the input features are the word vectors of a sequence of words from the reviews, and the predictors are the labels of the reviews.

The first step is to normalize and tokenize the data. First, I normalized all words into lower cases. Otherwise, some common words like "STUUNNING" in the review cannot be matched from the pre-trained word vector vocabulary, and then it will be recognized as an unknown word. Next, I replaced all "n't" into "not", and then I used "word_tokenize" from the Python package NLTK to tokenize sentences. I did this because words like "don't" would be tokenized into two tokens "do" and "n't", but "n't" cannot be matched from the pre-trained word vector vocabulary. Then I removed digits and punctuations as they do not help much for sentiment analysis.

I only used the first 30 useful tokens as inputs because people can know the sentiment of most reviews by reading the first 30 words, and thus, there is no need to waste computer memory for the remaining words. Then I used the pre-trained word vectors to transfer these tokens into certain IDs, and then I embedded their weights into the LSTM model. I transferred the labels into either 0 (negative) or 1 (positive) because the output of the LSTM model would be a number between 0 and 1. I also checked the counts of records with positive labels and negative labels to make sure that they are almost balanced for the training.

Figure 1 shows the structure of a LSTM model. There are three layers: an input layer, a hidden layer, and an output layer. The input layer is a 3-D tensor, and these dimensions are number of records by sequence length (here is 30) by word vectors (here is 300). The hidden layer is a LSTM layer contains 64 units with the dropout rate is 0.2 and the recurrent dropout rate is 0.2. The output layer is a 1-D layer that outputs a number between 0 and 1.

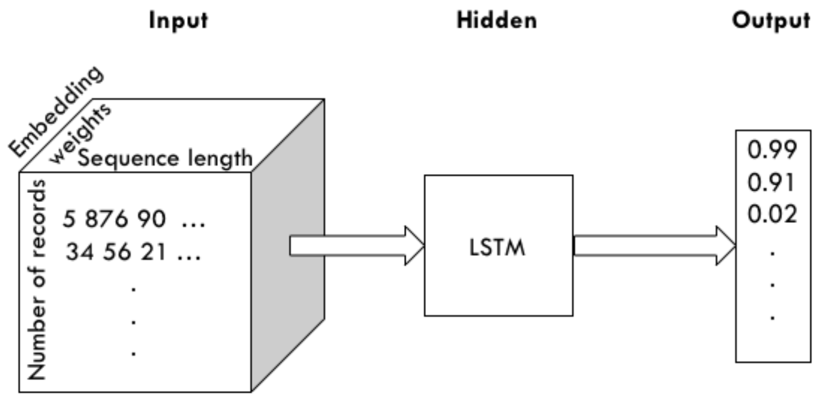

Figure 1. The three layers of the LSTM model

3.2 FastText classifiers

I used the Python package fasttext to build a fastText classifier. As the fastText classifier can automatically tokenize the text and generate its own word vectors, I did not do the steps like normalization, tokenization, removing digits and punctuations, limiting the length of the input sequence, and embedding as I did for building the LSTM models. As the fastText classifier is very efficient in computation, I used all 3.6 million reviews from the file "train.ft.txt" as the training dataset and all 400k reviews from the file "test.ft.txt" as the test dataset.

4.Results

Table 1 shows the results of the LSTM models and the fastText classifier. I used different numbers of records as inputs for the LSTM models. I only used three epochs for training for all the LSTM models, since adding more epochs for the training did not significantly increase or sometimes decrease the accuracy rate of the test data. As the limitations of my time and my machine, I built four LSTM models as table 1 shows. The accuracy rate of test data of these LSTM models increases as the size of training data increase. The best LSTM model I trained got a test accuracy rate of 0.88, and I used 160k reviews as the training dataset and 40k reviews as the test dataset for this model. For the fastText classifier, I got the test accuracy rate of 0.91, and I used 3600k reviews for training and 400k reviews for testing.

Method	Total records	Training data size	Test data size	Accuracy of test data
LSTM	10k	8k	2k	0.82
LSTM	50k	40k	10k	0.85
LSTM	100k	80k	20k	0.87
LSTM	200k	160k	40k	0.88
fastText	4000k	3600k	400k	0.91

Table 1. The results of the LSTM models and the fastText classifier.

Table 2 shows some manual test results of the best LSTM model with the test accuracy rate of 0.88 and the fastText classifier with the test accuracy rate of 0.91. For the LSTM model, in theory the output value of a negative review should be very close to 0, and the output value of a positive review should be very close to 1. So, I set the output sentiment as "negative" when the output value is lower than 0.2, set the output sentiment as "positive" when the output value is larger than 0.8, and set the output sentiment as "neutral" when the output value is between 0.2 and 0.8. With this setting, table 2 shows that the output results of the best LSTM model are quite reasonable in the real world. For the fastText classifier, the output format is different that it outputs a label and a probability with that label. Table 2 shows that the fastText classifier gives the same sentiment predictions for the positive and negative samples as the best LSTM model does. However, the fastText classifier gives the negative labels with the probabilities of 0.96 and 0.80 for the last two neutral samples in table 2, which means it cannot identify the neutral samples as the best LSTM model does.

Input test sentence	LSTM output	LSTM sentiment	fastText output (label, prob)	fastText sentiment
This is a joke	0.00744833	Negative	('1', 0.998047)	Negative
The worst product I have ever bought	0.00443599	Negative	('1', 1.0)	Negative
Complete waste of money.	0.00449477	Negative	('1', 1.0)	Negative
I like it, and I want to buy it again.	0.97385174	Positive	('2', 0.740234)	Positive
This is the best 30 bucks that I have ever spent	0.9875195	Positive	('2', 0.923828)	Positive
My mom loves it.	0.99224257	Positive	('2', 1.0)	Positive
I am going to give a presentation.	0.52989006	Neutral	('1', 0.962891)	Negative
I am going to have an exam.	0.71282566	Neutral	('1', 0.802734)	Negative

Table 2. The result of manual tests

5. Conclusions

This paper uses LSTM models and a fastText classifier to predict the sentiment of a sequence of words. For the LSTM models, the accuracy rate of test data increases as the size of training data increase. The best LSTM model I trained with the test accuracy rate of 0.88 is sufficient for real-world applications as the manual test results are quite reasonable in life. For the fastText classifier, it is much more efficient in computation and much easier to build than the LSTM models since it can automatically tokenize the text and generate its own word vectors. However, the fastText classifier I trained in this paper cannot identify neutral sentences as the best LSTM model does.

Sources or references:

[1] Amazon product reviews data: https://www.kaggle.com/bittlingmayer/amazonreviews/data

[2] Pre-trained word vectors from Google News dataset: https://code.google.com/archive/p/word2vec/#Pre-trained_word_and_phrase_vectors

[3] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of Tricks for Efficient Text Classification

[4] Python package fasttext: https://pypi.python.org/pypi/fasttext

[5] Python package keras: https://keras.io/