Sentiment analysis on 'Consumer Reviews of Amazon Products' dataset

Bodonyi Dániel **Gonzalez Guerra Rebeca Sarai** **Kooshkjalali Navid**

Abstract

Sentiment analysis is a method of categorizing opinions expressed in a text. It is an active field of research in NLP. In the following, a long short term memory (LSTM) based recurrent neural network is developed on Adam Bittlingmayer's 'Amazon Reviews for Sentiment Analysis' dataset and evaluated in different ways such as F1 score, precision, recall and confusion matrix, it is then compared to other works on the same dataset.

1 Introduction

In the following, the datasets used are introduced, efforts in balancing and processing the data are presented. Next an LSTM based model is built and trained on the data. The performance characteristics of the model are calculated and compared with other works on the same dataset. Lastly some other machine learning approaches we experimented with are presented.

2 Dataset

Initially we used a much smaller dataset: the 'Consumer Reviews of Amazon Products' dataset by Datafiniti. This dataset contains only 34000 reviews as well as a rating from 1 to 5. This dataset is very unbalanced with overwhelmingly positive reviews. When trying to use this dataset we had to complement it with yet another dataset to add more negative or neutral reviews. The results of our model training on this dataset were sub-par, and for this reason we decided to use a larger dataset.

The chosen dataset is the 'Amazon Reviews for Sentiment Analysis' dataset provided by Adam Bittlingmayer containing a few million Amazon reviews in fastText format. The training set for this data set is quite large (about 500 MB), so we decided to only use the test set. The test set contains 400000 reviews (50 MB), out of which only 200000 were used. All reviews of this dataset are in English. The dataset contains the raw strings of the review as bytes and is labelled as either Positive or Negative.

3 Data or text augmentation

When handling the original dataset, we experimented with text augmentation. Data augmentation is a strategy that enables practitioners to significantly increase the diversity of data available for training models, without actually collecting new data by applying transformations to the original labeled examples to construct new data for the training set. It has been exceedingly successful with images, where models can be invariant to mirror images and rotations by generating synthetic/augmented data from mirroring or rotating existing images and having the model train on the data. Some examples of such transformations that are relevant in NLP include back-translations, conditional BERT, synonym replacement, random deletion, etc. In the end we did not need to use data augmentation as we swapped to a larger dataset.

4 Balancing of the dataset

Initially, The first dataset was manually balanced over the label classes by up-sampling/over-sampling the minority class or down-sampling/under-sampling the majority class using re-sampling (bootstrapping) techniques. The dataset 'Amazon reviews US Electronics' by Janet Kelly was used to bring in more negative and neutral data. Using a combinations of both dataset a fully balanced dataset with almost 25000 entries per each sentiment (neutral, negative, positive) was generated.

The final dataset we used was already balanced, with only a few percent difference between negative and positive reviews, with no missing values. Balancing this dataset was as easy as dropping the few extra positive reviews, so that

finally, the balanced dataset contains 99435 positive and negative reviews each.

5 Pre-processing the data

The review data needs to be processed and cleaned. As the text is stored in binary format, it first has to decoded to a UTF-8 string. The next pre-processing steps consists of lower-casing and then tockenizing the text, removing URLs, removing any numbers, special characters and trailing whitespace, eliminating the stopwords from the tokens. This process changes the distribution of sentence length.

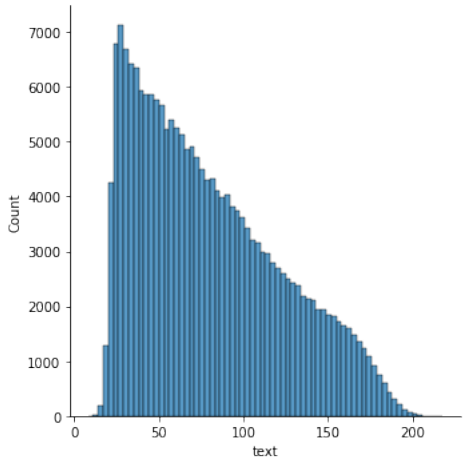

Figure 1: Sentence length distribution before balancing

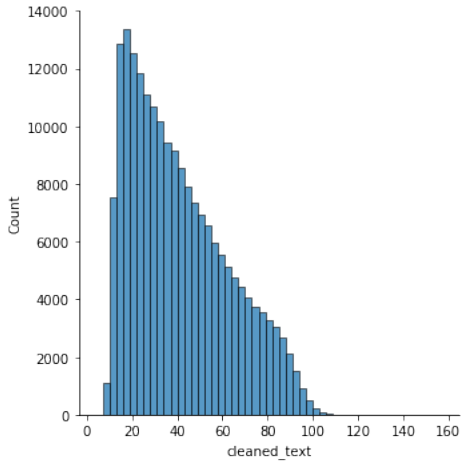

Figure 2: Sentence length distribution after balancing

6 Creating Vocabulary

Next a vocabulary is created by extracting the 1000 most common words from the text using a Counter. Tokens for padding and unknown words are also added to the vocabulary.

It's worth noting that after we trained the model and tried to test it, a very large percentage of words were assigned the unknown token, this means the size of our vocabulary was too small and we should have taken more of the most common words.
After this step we can split the data to train and test sets.

7 Padding

Initially we used post padding for the input (All the sequences were padded with zeroes in the ending according to the longest sequence's length or a chosen length longer than the longest length.) This padding lowered the accuracy of our model significantly. The accuracy improved significantly when we used pre-padding (The sequences are padded with zeroes in the beginning). An explanation for this regarding RNNs is that, post-padding seems to add noise to what has been learned from the sequence through time, and there aren't more timesteps for the RNN to recover from this noise. With pre-padding, however, the RNN is better able to adjust to the added noise of zeros at the beginning as it learns from the sequence through time (1).

In hindsight, it would have been much better to utilize Pytorch's PackedSequence when dealing with variable sequence lengths.

8 Model

The model we used for consisted of a simple embedding layer with a dimension of 400, connecting to a single-directional LSTM, with 2 layers and 512 features in the hidden state. The LSTM connects to a linear fully connected layer, after which a sigmoid activation function is applied. A dropout of 20% was used.
The model is saved if the test accuracy and loss improve during training.

9 Performance

The model's performance is evaluated in a number of ways. The following confusion matrix was created from the predicted and gold values of the test data. The rows and columns represents the labels of data, 0 is negative sentiment and 1 is positive sentiment. The main diagonal represents the true positive and true negatives and represent when the model made a correct prediction about the sentiment of a sentence. Top right represents reviews where the model predicted a positive

sentiment and in reality the review was negative and the bottom left represents reviews where the model predicted a negative sentiment and in reality the review was positive.

$$\text{Confusion} = \begin{bmatrix} 22078 & 288 \\ 2290 & 22411 \end{bmatrix}$$

The classification report results of the training set is the following:

	precision	recall	F1-score	support
positive	0.8860	0.9074	0.8966	24698
negative	0.9062	0.8845	0.8952	24966
accuracy			0.8959	49664
macro avg	0.8961	0.8960	0.8959	49664
weighted avg	0.8962	0.8959	0.8959	49664

10 Comparison

We can check how other publicly available models have performed on the same dataset.
A very similar approach, with a bidirectional LSTM reached 96% accuracy.

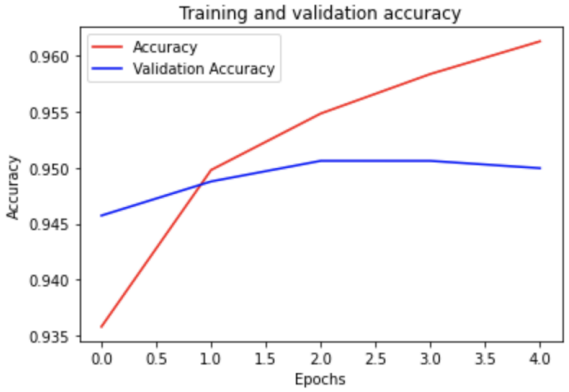

Figure 3: Fully connected bidirectional LSTM

Another LSTM / Gated recurrent unit approach reached 93.9% accuracy. A fastText supervised model with 91.7% accuracy, and a DistilBert approach with 90.3% accuracy.

11 Machine Learning Models

Other than the LSTM based model, we also tried some machine learning models. Namely, a random forrest classifier and a naive Bayes approach.

12 TF-IDF Features

Term Frequency–Inverse Document Frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. We use the sklearn TF-IDF vectorizer to extract these features from the review texts.

12.1 Term Frequency (TF)

The number of times a word appears in a document divided by the total number of words in the document. Every document has its own term frequency.

12.2 Inverse Document Frequency (IDF)

The log of the number of documents divided by the number of documents that contain the word w. Inverse data frequency determines the weight of rare words across all documents in the corpus.

13 Classifying reviews

Two models of classification are explored and their performance is compared.

13.1 Random Forest

Random forest consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the forest has its own class prediction and and the mode of all predictions becomes the model's prediction. After pre-processing the data, the TfidfVectorizer is used to create the input. A random forest with 100 estimators was used and accuracy, precision, recall, F1-score and the confusion matrix was measured.

$$\text{Confusion} = \begin{bmatrix} 4482 & 428 & 152 \\ 370 & 4333 & 444 \\ 95 & 333 & 4590 \end{bmatrix}$$

	precision	recall	F1-score	support
negative	0.91	0.89	0.90	5062
neutral	0.85	0.84	0.85	5147
positive	0.89	0.91	0.90	5018
accuracy			0.88	15227
macro avg	0.88	0.88	0.88	15227
weighted avg	0.88	0.88	0.88	15227

Accuracy = 0.880344

13.2 Naive Bayes

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem, where every pair of features being classified is independent of each other. This assumes that that words in review text are independent and appear

randomly. Despite this assumption they have great performance in classification. A Count Vectorizer and then a TF Vectorizer is applied on training data. For the naive Bayes model MultinominalNB is used. 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.

$$\text{Confusion} = \begin{bmatrix} 4307 & 603 & 152 \\ 184 & 5525 & 438 \\ 102 & 311 & 4605 \end{bmatrix}$$

	precision	recall	F1-score	support
negative	0.94	0.85	0.89	5062
neutral	0.83	0.88	0.85	5147
positive	0.89	0.92	0.90	5018
accuracy			0.88	15227
macro avg	0.89	0.88	0.88	15227
weighted avg	0.89	0.88	0.88	15227

Accuracy = 0.882445

As observed the Naive Bayes model has slightly improved performance over the Random Forest model.

13.3 References

N V Subba Reddy, N V Subba Reddy, *EFFECTS OF PADDING ON LSTMS AND CNNS*, 2019.