Movie Review Sentiment Analysis

Delong Meng

Nov 26th, 2021

Introduction

Natural language processing (NLP) is a specialized field in machine learning. The critical step in a NLP task is to convert natural language text into the format of a vector or matrix so it can be fed into a machine learning model for training. In this project, we are provided with movie reviews including the review text and the corresponding sentiment (positive or negative). This is a typical sentiment analysis problem and the goal is to build a classification model to predict the sentiment based on movie reviews.

Methods

Train/test splitting of the data

This dataset contains 50,000 movie reviews in total. For each review entity, we have the sample ID, the review text, the score, and the sentiment. The score is based on a 10-point scale, with scores 1-4 corresponding to negative sentiment, and scores 7-10 corresponding to positive sentiment. This dataset contains no reviews of score 5 or 6. Sentiment can be either 0 (negative) or 1 (positive). Note that in the training and prediction of this project, we only use the information of the review text as predictor and the sentiment as the response.

To train the model and evaluate the performance, we randomly split the dataset into 50% training set (25,000 reviews) and 50% test set (25,000 reviews). Five such splits were generated according to pre-defined test IDs.

Vocabulary construction

To train a model, we need to first extract features from the review text. For example, if we have a vocabulary of words, we can simply count the frequency of all of the words that appear in a given review text. Then, we can use this information as the predictors for the model training. This is called "bag-of-words model". In addition to individual words, we also consider phrases, or sequences of n contiguous words, also called "n-gram". Considering both individual words and n-grams, the total number of possible terms in the vocabulary extracted from our dataset (all 50,000 movie reviews) will be huge, limiting the computation time and model interpretability. Thus, we attempted to reduce the size of vocabulary by only keeping the most important terms in the vocabulary. Finally, we reduced the vocabulary to be under 1000 terms. Details about the vocabulary construction can be found in a separate RMarkdown file. Briefly, we cleaned the text data first by removing the HTML tags, defined a list of "stop words", used the `text2vec` R package to create and prune the initial vocabulary, and built the document-term matrix (DTM). Then we utilized logistic regression with Lasso regularization to further trim the vocabulary by limiting the number of non-zero features in the model.

Text embedding and modeling

Input review text will be processed similarly to the steps in vocabulary construction using the `text2vec` R package. The text will first be cleaned by removing the HTML tags. Then, an iterator over the tokens will be generated for the review text after the preprocessing of converting characters to lower case. Next, a vectorizer will be created based on the vocabulary that we constructed before, including both the 1-grams (individual terms) and 2-grams of the vocabulary. Finally, a DTM is created using the token iterator and the vectorizer, so that one piece of review text is now one row of features in this matrix.

For the modeling part, a cross-validation (10-fold by default) logistic regression model with Ridge regularization will be fit for the training data using the `cv.glmnet` function from the `glmnet` R package. The sentiment (positive or negative) will be used as the response. Review text in the test set will be processed the same way as the training data, and the trained model will be applied to the test data with the `lambda.min` from the cross-validation as the lambda value to predict the probability of this given review text being positive.

Performance evaluation

As mentioned above, logistic regression models will be built to predict the probability of the sentiment of a given review being positive. A series of threshold values can be used as cutoffs to classify a review to be positive or negative based on the predicted probability. Thus, a specificity/sensitivity curve can be generated for a model, and the area under the curve (AUC) can be used as a metric to evaluate the performance of the model. In this project, we use the `roc` and `auc` functions from the `pROC` R package to calculate the AUC.

Results

Performance of the sentiment prediction model

We first generated an initial vocabulary using the `text2vec` R package and all of the review text from the whole dataset, and reduced the vocabulary to the size of 976 using a logistic regression model with Lasso regularization. Review text in each training set was vectorized using this relatively small vocabulary, and then logistic regression model with Ridge regularization was trained and tested in the corresponding test dataset. Overall, this strategy had great prediction performance based on the AUC metric. As shown in Table 1, the AUC exceeded 0.96 for all of the 5 splits of train/test dataset. Figure 1 shows the ROC plot of one of the train/test split. The ROC describes how the specificity and sensitivity change according to different thresholds used to convert the probability to classification. Thus, the AUC metric considers both false negatives and false positives. Taken train/test split 1 as an example (Figure 1), if we use 0.5 as a cutoff, then the specificity and sensitivity are 0.90 and 0.92, respectively.

Running time and computer system

Running time for each split of train and test datasets is also shown in Table 1. The computer system used in this project was: MacBook pro, 2.2 GHz, 16 GB memory.

Table 1. AUC and Runtime for each split

Train/Test split	AUC	Runtime (seconds)
1	0.9684	45.30
2	0.9682	37.27
3	0.9681	37.69
4	0.9690	36.42
5	0.9676	42.64

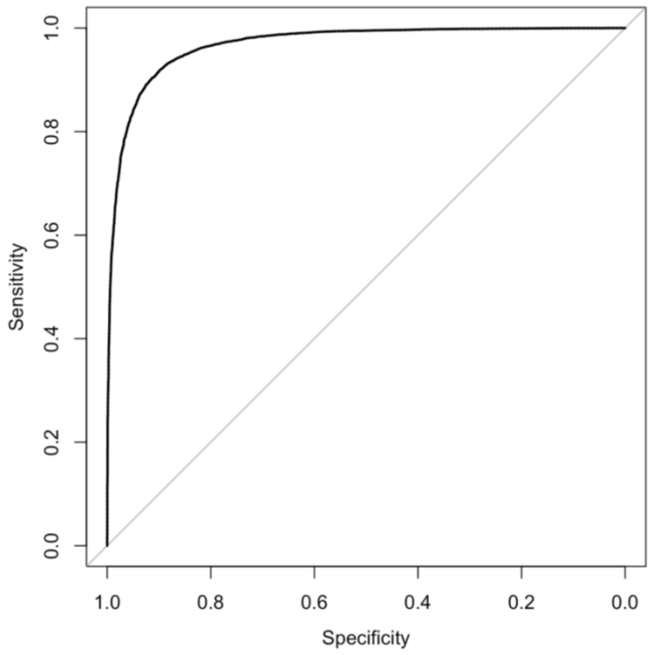

Figure 1. ROC plot of train/test split 1

Discussion

Model performance

Overall, the model performs pretty good. This is impressive since the “bag-of-words” algorithm is relatively simple. We can take a look at true positive, true negative, false positive and false negative examples to better understand how the model makes the predictions.

Interpretability

Since the model is simple enough, the prediction is very interpretable. We can easily extract the coefficients from the trained model. For a given review text (see the example in Table 2), we can see all of the terms it contains and their counts from the DTM and corresponding coefficients from the model. The most positive terms and the most negative terms in this example are highlighted in red and blue, respectively. A linear regression score (eta) is simply calculated based on the listed information and then converted to a probability (p). If we use 0.5 as a cutoff here, the model will classify this review as positive, which is a true positive.

Table 2. An example showing the modeling procedure from review text to sentiment prediction

Review (id: 12782; sentiment=1, score=10)	Term	Coefficient	Count
"This movie about a group of small town teens that decide to rob the local bank is excellent. Brian (Justin Walker) wants to get out of his small town, much like Jimmy Stewart in \"It's A Wonderful Life.\" However, unlike George Bailey, Brian is going to rob a bank to finance his dream of attending art school, even if his father is not supportive. The offer to Brian is to act like a customer and distract the guard. It's a tempting offer that if offered to many, I question what they would do. Anyways, Brian does it. When the Sheriff (James Remar) and his force surround the bank, things go from bad to worse. It's a standoff with even the Feds moving in to kill the kids if they have a clean shot. The Sheriff must prevent this and try to end the standoff in a peaceful way. Unfortunately, tensions rise, and the teens inside turn on each other. Some are out of control. The paper cutter scene is gruesome and hard to watch. Very intense!"	excellent	0.60603069	1
	wonderful	0.57346654	1
	intense	0.35659816	1
	unlike	0.22512912	1
	moving	0.18886903	1
	each	0.17023045	1
	small	0.13331252	2
	life	0.12157021	1
	very	0.09847549	1
	it's	0.05995484	3
	many	0.02475314	1
	would	-0.0956165	1
	even	-0.1186378	2
	bad	-0.3606225	1
	worse	-0.5031541	1
	unfortunately	-0.5108017	1
	(Intercept)	*0.0606168*	

eta = 1.165, p = 0.762, prediction = positive

Thus, for any given review text, we can perform similar analysis and understand why the model predicts it as positive or negative. From the above example, we can see some interesting things. For instance, the word "wonderful" is a very positive term (which makes sense), however in this context, it shows up within the name of another movie called "It's A *Wonderful* Life", instead of being a comment to a movie.

Limitations

We can also perform similar analysis to misclassified review texts and learn how the model fails in certain situations. One obvious limitation of the "bag-of-words" model is that it only counts the frequency of words but ignores the order of words (or sentences). For example, the features extracted from "A is better than B" will be exactly the same with "B is better than A", although they have totally opposite meaning. In addition, the context is important. For example, some reviews use good words to comment a different movie instead of the current one (something like in a review for movie A it mentions "movie B is great"), in this case, it doesn't make sense to include those positive words to predict the sentiment towards this movie. Furthermore, sometimes people use sarcasm when expressing their opinions, which is obviously often misleading the model.

More sophisticated models could be used to further improve the performance of our model. For example, deep learning models are commonly used in NLP tasks. Specifically, recurrent neural networks (RNNs) consider the order of the sequence, and bidirectional RNNs with long short-term memory (LSTM) further take into account the context information. "Skip-grams" can also be used to be more flexible to capture related words.

Acknowledgement

Dr. Feng Liang's instructions