Sentiment analysis with R: Predicting customer ratings of Amazon Kindle books from text reviews

Code available in https://github.com/pablo-crucera/sentiment-analysis

Introduction

Natural Language Processing is a subfield of several disciplines, such as linguistics, computer science and artificial intelligence, that aims to provide machines the possibility for "understanding" human language. This is done through a set of techniques that have experienced unbelievable improvements in the very last few years thanks to the irruption of new neural architectures, especially transformers.

The state-of-the-art models within this field are capable of yielding outcomes that resemble human-like behavior not only in translation, but also in many other tasks such as the one that we will carry out in this work: sentiment analysis.

Sentiment analysis refers to the field of text mining whose goal is to extract subjective knowledge from word sequences. In this work, we will try to employ sentiment analysis to perform text classification over a set of text reviews on a product by trying to predict their labels (the number of stars given to the product).

Problem description

We have a dataset containing 12,000 text reviews of books extracted from Amazon Kindle Store obtained from Kaggle which has been already preprocessed. It consists of four columns:

- #: the review identifier in the dataset.
- rating: the numerical rating given by the reviewer, integer in the interval [1, 5].
- summary: the summary/title of the review.
- reviewText: the text body of the review.

The objective of this work is to create a classification model able to predict the numerical rating given to a book after providing its review text by applying NLP techniques.

Methodology

We will work in R language using RStudio.

Basic checks

The first thing that we will do is to perform a set of basic checks to make sure that the text is correctly written: all characters are valid and normalized in the UTF-8 codification for English. Another check is to ensure that the dataset is correctly balanced in number of classes, which will be done through observing a histogram and subsampling if necessary.

Word tokenization

After that, a tokenization will take place: every word will be converted to an integer according to its frequency in all sentences, that is, 1 will be assigned to the most frequent word, 2 to the second one and so on. We will define a maximum number of features (left as a tunable parameter by the user) used by the tokenizer, which will be the length of our vocabulary. This means that if we select *x* as the maximum number of features, after the *x*-th most frequent word, all words will be considered to be the same (unknown) and will be assigned token 0. This is done for both reducing computational costs and improving the results: it might not make any sense to consider words whose meaning the machine may not understand because of their scarce appearance in the examples.

Method 1: LSTM + dense layer

Word embedding

After that, word embedding occurs: each token (word) is projected into a space of much higher dimensionality. The goal of this projection is to keep similar words (syntactically, semantically…) closer in the output space and is done by a neural network.

The learning process of the word embedding is carried out by considering each word's *n*-skip-gram: what the *n* closest neighboring words to that word are, both back and forth.

Word embedding leaves us then 6 tunable parameters: the dimension of the output vector, the skip-gram number, the number of negative samples (that represents the number of words for which we want to update the weights every time that a token enters the network, to prevent the whole network from being updated), the number of epochs, the number of steps per epoch and the batch size.

Splits generation

Since we have 5 different discrete levels that we will consider as the class labels, we will perform one-hot encoding over the rating column by creating dummy variables. After that, the data is randomly splitted into test and training sets, with a probability of assigning an instance to the test split that is provided by the user (the rest of the instances are assigned to the train sets). Then, review texts are defined as the input to the model(s) and their respective one-hot encoded ratings as the class labels.

Classifier LSTM + dense layer

First, there is an embedding layer (which can be learnt from 0 or may employ the embedding weights learnt before), followed by an LSTM (long-short term memory) layer, which is a recurrent neural network architecture that is capable of "remembering" past sequences. Its output is then fed to a dense layer with five different output neurons, each of them representing one different class. The tested activation function at the end is softmax. In between layers, there is a dropout layer with rate set to 0.5 to prevent the model from overfitting.

Tuneable parameters: number of LSTM units (outputs of the LSTM architecture), dropout of the LSTM layer and whether the embedding layer is learnt from 0 or uses the previous model weights.

Method 2: Multinomial Naïve Bayes

In this classification method, the number of maximum features is unbounded because a great dictionary does not represent any unaffordable computational expense.

After tokenization, a document-feature matrix is created for the training split. In the case of the test set this is done too but forcing features to match those of the training set. Then, the model is built making the prior probabilities of each term follow a probability distribution proportional to their frequency.

Tuneable parameters: smoothing parameter for feature counts by class.

Results and discussions

All checks showed that the data was correctly preprocessed and was usable.

LSTM-based model

In all tested executions, the best obtained validation accuracy was around 48-50% and became easily overfitted after a short number of epochs regardless of the parameters used. Execution times range from ~30 seconds to ~8 hours with little difference in results. For instance, a *simple* model with a dictionary containing 2000 words, maximum length of 1500 words, embedding size of 32 and 64 LSTM units, with integrated embedding, using a test set with 20% of all instances produces the following outcomes (model that has shown the least overfitting):

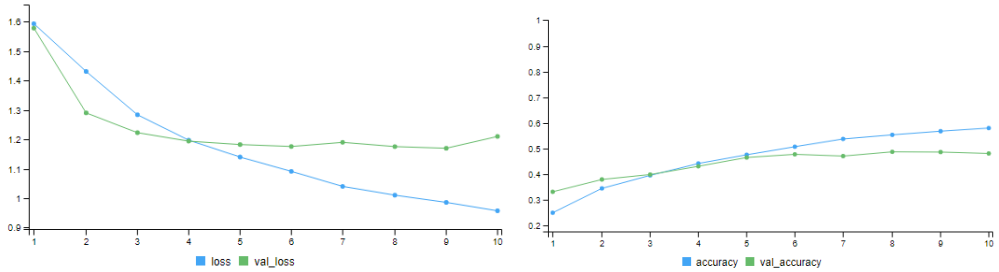

Figure 1: loss functions (left) and accuracies (right) shown during the training process of a LSTM-based model

```
Confusion matrix:
          Reference
Prediction   1   2   3   4   5
         1 201 112  35  24  19
         2 121 159 106  37   9
         3  19  57  96  72  23
         4  14  22  92 211  73
         5  16  25  57 286 465

Metrics by class:
         Sensitivity Specificity Pos Pred Value Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate Detection Prevalence
Class: 1   0.5417790   0.9040404      0.5140665      0.9132653 0.5140665 0.5417790 0.5275591  0.1578052      0.08549553            0.1663122
Class: 2   0.4240000   0.8618421      0.3680556      0.8874414 0.3680556 0.4240000 0.3940520  0.1595066      0.06763080            0.1837516
Class: 3   0.2487047   0.9129771      0.3595506      0.8608445 0.3595506 0.2487047 0.2940276  0.1641855      0.04083369            0.1135687
Class: 4   0.3349206   0.8832074      0.5121359      0.7839092 0.5121359 0.3349206 0.4049904  0.2679711      0.08974904            0.1752446
Class: 5   0.7894737   0.7820658      0.5477032      0.9174434 0.5477032 0.7894737 0.6467316  0.2505317      0.19778818            0.3611229
         Balanced Accuracy
Class: 1         0.7229097
Class: 2         0.6429211
Class: 3         0.5808409
Class: 4         0.6090640
Class: 5         0.7857698

Overall metrics:
     Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue  McnemarPValue
  4.814972e-01   3.423070e-01   4.611209e-01   5.019197e-01   2.679711e-01  8.154303e-108   8.451989e-33
```

Figure 2: validation metrics obtained by the LSTM-based model over the test set

The model shows a high specificity and a balanced accuracy for all classes that in the worst case is over 58%. By looking at the confusion matrix, one can observe that mislabeled ratings are mostly given in cases where the predicted value is a *neighbor* of the real label, that is,

misclassified real 1-star ratings are mainly predicted to be 2, mislabeled 4-star ratings were mostly believed to be 5. The real label for which worst performance is observed is 3, which represents mediocre scores. This statement is reinforced by the fact that if we addressed this problem as a regression problem, we would have a RMSE of 1.10.

What can be inferred from here is that the model is mostly able to distinguish between *good*, *average* and *bad* ratings, even if the predicted label does not correspond to the real class.

Multinomial Naïve Bayes model

Execution times drop drastically for this model, which takes around 1 or 2 seconds to be executed. The validation results for a model using 1 as the smoothing parameter and using 20% of the instances for the test set are shown below.

```
Confusion matrix:
          Reference
Prediction   1   2   3   4   5
         1 203  80  21  14   6
         2 105 191  80  31   7
         3  21  63 112  79  20
         4  32  69 135 360 200
         5  13  18  28 138 353

Metrics by class:
         Sensitivity Specificity Pos Pred Value Neg Pred Value Precision    Recall        F1 Prevalence Detection Rate Detection Prevalence
Class: 1   0.5427807   0.9396509       0.6265432       0.9167883 0.6265432 0.5427807 0.5816619  0.1572089      0.08532997             0.1361917
Class: 2   0.4536817   0.8861083       0.4613527       0.8829517 0.4613527 0.4536817 0.4574850  0.1769651      0.08028583             0.1740227
Class: 3   0.2978723   0.9086370       0.3796610       0.8733205 0.3796610 0.2978723 0.3338301  0.1580496      0.04707860             0.1240017
Class: 4   0.5787781   0.7518497       0.4522613       0.8344915 0.4522613 0.5787781 0.5077574  0.2614544      0.15132409             0.3345944
Class: 5   0.6023891   0.8901283       0.6418182       0.8726080 0.6418182 0.6023891 0.6214789  0.2463220      0.14838167             0.2311896
         Balanced Accuracy
Class: 1         0.7412158
Class: 2         0.6698950
Class: 3         0.6032547
Class: 4         0.6653139
Class: 5         0.7462587

Overall metrics:
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue  McnemarPValue
  5.124002e-01   3.778764e-01   4.920995e-01   5.326703e-01   2.614544e-01  2.227445e-149   1.754334e-09
```

Figure 3: validation metrics obtained by the Multinomial Naïve Bayes model over the test set

Not only execution times improve, but also results (although not in a much significant manner). Misclassifications are also mostly given in cases in which a similar rating to the reference value has been assigned. Again, the worst performance is shown in cases where the true label is 3. Recovering the numerical nature of the data, the rooted mean squared error is now 1.05.

Recalling the conclusions drawn from the other model, we can add a new one: a very simple model (Multinomial Naïve Bayes) has been able to outperform a much more complex one (a recurrent neural network with embedding) in terms of accuracy, F_1-score and training time.

Conclusions

In short:

- In general, both models have been able to detect the positivity or negativity of a review.
- A simple model sometimes may be a better choice than a very complex one.

References

https://www.kaggle.com/meetnagadia/amazon-kindle-book-review-for-sentiment-analysis

https://blog.cambridgespark.com/tutorial-build-your-own-embedding-and-use-it-in-a-neural-network-e9cde4a81296

M. Rico, “NLP Hands-On”. Course slides in subject Intelligent systems, in Máster Universitario en Ciencia de Datos, Universidad Politécnica de Madrid.

https://rpubs.com/nabiilahardini/word2vec