Movie Sentiment Classification

Stefan-Cristian Roata

16 November 2021

Introduction

In this model report, I will be talking about my implementation of sentiment classification on the NLTK movie reviews dataset and what led me to it. I will also be talking about performance metrics such as accuracy, confusion matrix, area under the ROC curve and execution time. In the end, I will contrast the performance of my chosen model against another classifier that I tried in parallel. Let us now start discussing the best model I have managed to train:

1 Logistic Regression

One of the simplest models available to us is regression. I first learned about the different kinds of regression (linear, logistic, Poisson etc) last semester when I took the *Introduction to Data Science* course. What I remembered from that course is that, in linear regression modelling, the outcome variable is usually continuous, but if we are more interested in a categorical outcome such as yes/no, pass/fail, positive/negative, then Logistic Regression might be used. Once applied, this kind of model will provide us with the probability p that a particular review in our dataset is positive, and based on a certain threshold (which is usually 0.5), will continue to classify each review in either of the two categories: negative (when the probability is less than the threshold) or positive (when the probability is higher than the threshold). This is perhaps less than fully accurate, as what we are actually trying to model is the log-odds ratio $\ln \frac{p}{1-p}$ as a linear function of the features $x_1, x_2, \ldots x_n$ (which in our case are the words and their corresponding TF-IDF scores in the documents):

$$\ln \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n.$$

The package that I used to build the model is *sklearn*, quite popular in the Python machine learning community for its efficiency and ease of use. The model building pipeline that I followed is the following:

1. read the reviews data and preprocess it by tokenizing on spaces, removing stop words and punctuation, followed by stemming and re-adjoining the list of words generated via this process;

2. create the TF-IDF matrix using the TfIdfVectorizer from sklearn, which converts the raw text data into a much nicer format to work with:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(texts)
Y_train = labels
```

   I initially wanted not to train on the whole *movie_reviews* data and do a classic "80-20" train-test split, but then I realized that this approach would not make sense as I already have the test data that was provided (the raw texts in the /small/... folder). Therefore, I trained my model on the whole *movie_reviews* dataset.

3. instantiate a LogisticRegression model object and fit the vectorized movie reviews training data to it:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
```

The last part was to find how accurate the model was:

2 Prediction Performance

2.1 Initial overview

Without calculating any accuracy measure yet, does our model pass face-value? That is, can it predict something quite simple (a review that is clearly negative or clearly positive) correctly? Let's see...

```
print('Result for a negative:', predict_sentiment("This is a bad movie.
The worst I even seen."))
print('Result for a positive:', predict_sentiment("Best movie ever.
Wonderful. Love it."))
# Result for a negative: neg
# Result for a positive: pos
```

Our model seems to be able to classify obviously negative/positive reviews correctly. That is a good start. Now on to accuracy metrics:

2.2 Accuracy Metrics

One first thing I did was to see how well the model fit the initial training data:

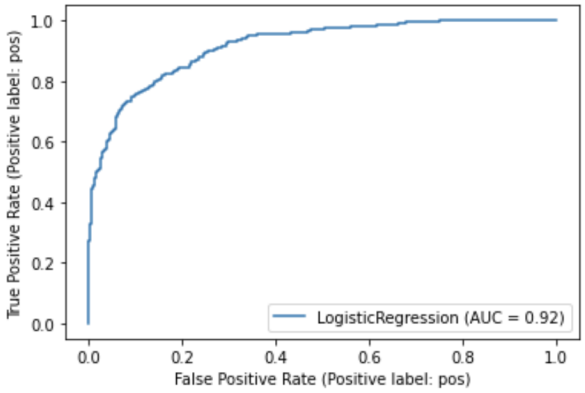
```
print(logreg.score(X_train, Y_train))
# 0.9735
```

An accuracy score of 97.35% sounds amazing! Nevertheless, this is the accuracy score on the data that was used to train the model, so of course we would expect that to be high. At first, I thought this score was a bit "too high", as I feared overfitting of the training data, but once I calculated the accuracy score on the test data, I was relieved:

```
print(logreg.score(X_test, Y_test))
# 0.829
```

An accuracy score of 82.9% on the test data is a bit lower than the stellar score of 97.35% that we obtained earlier, but it's still quite decent, as the model has never seen this new data before. The risk of having overfitted to the training data now seems quite small.

Besides calculating the accuracy score, I also computed the receiver operating characteristic curve (ROC) and the area under it:



In general, an Area under the Curve (AUC) close to 1 denotes that the model is quite good at classifying the data. Our AUC for Logistic Regression is 0.92 in this case, which is a sign that our model is performing great. The shape of the ROC curve also confirms this, as we can see it is quite steep. Finally, I computed the confusion matrix to see exactly how many true positives, true negatives, false positives and false negatives I got:

```
predictions = logreg.predict(X_test)
confusion_matrix(Y_test, predictions)
# array([[417,  83],
#        [ 88, 412]])
```

		PREDICTED		
		Negative	Positive	Total
ACTUAL	Negative	417	83	500
	Positive	88	412	500
	Total	505	495	1000

The number of true positives and true negatives (412 and 417, respectively) is quite good, while the number of false negatives and false positives is low. This is another indicator that our model is good. Nevertheless, there are more false negatives (88) than false positives (83), so the model tends to classify more reviews as negative even if some of them are actually positive.

2.3 Execution Time

The time that the model takes to classify exactly 1000 reviews is quite short (about 3.51s per loop):

```
%%timeit
for x in list(dataset['text'])[:1000] :
    predict_sentiment(x)
# 3.51 s ± 122 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

The time it takes for the script to load via *import* is also short:

```
import time
start = time.time()
import sentiment_predict as student
medium = time.time()
print("Import:", medium - start)
# Import: 8.702278137207031e-05
```

3 Comparison with another model: Multinomial Naive Bayes

The other model that I tried was Multinomial Naive Bayes. This classifier is suitable for classification with text-derived features (e.g. word counts for text classification), which is exactly what we are dealing with here. I implemented it similarly to the Logistic Regression classifier using sklearn:

```
from sklearn.naive_bayes import MultinomialNB
multinomial_nb = MultinomialNB()
multinomial_nb.fit(X_train, Y_train)
```

The Multinomial Naive Bayes Classifier was trained on the exact same dataset as the Logistic Regression (i.e. all movie reviews).

I tested this new classifier in a similar way to the previous model. First of all, I tried to see if it passes the face value judgment, and it definitely does: it can classify a clearly negative review as negative and a clearly positive review as positive:

```
print('Result for a negative:', predict_sentiment_NB("This is a
bad movie.The worst I even seen."))
print('Result for a positive:', predict_sentiment_NB("Best movie
```

```
ever.Wonderful. Love it."))
# Result for a negative: neg
# Result for a positive: pos
```

Looking at the accuracy on the training data, we would expect it to be quite high (after all, the same data was used for learning), and it is this:

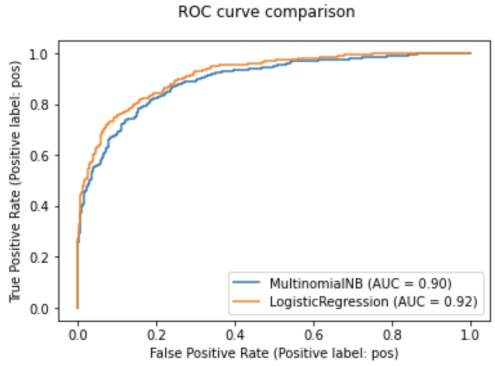
```
print(multinomial_nb.score(X_train, Y_train))
# 0.9595
```

Nevertheless, this accuracy score is a little lower than that obtained by the Logistic Regression model. This does not matter as much as the accuracy this new classifier obtains on the test data:

```
print(multinomial_nb.score(X_test,Y_test))
# 0.808
```

Here, we can actually see a significant difference from the previous model. While Logistic Regression obtained a score of 82.9% on the test data, Multinomial Naive Bayes only obtained 80.8%. This is one of the reasons why I chose Logistic Regression over Naive Bayes as my main model to deploy.

We can also observe how the two models compare in terms of their ROC curves:



The Multinomial Naive Bayes classifier has an area under the curve of only 0.90 as compared to the Logistic Regression classifier that has 0.92. It can be concluded that Logistic Regression is the better model, albeit only incrementally better.

In terms of the time that it took to classify 1000 reviews, Multinomial Naive Bayes was just 0.1 seconds slower on average than Logistic Regression:

```
%%timeit
for x in list(dataset['text'])[:1000] :
    predict_sentiment_NB(x)
# 3.61 s ± 23.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Conclusion

Logistic Regression has performed better overall as compared to the Multinomial Naive Bayes classifier, surpassing it in every aspect: train data accuracy, test data accuracy, AUC and execution time. There are perhaps even better performing models for the task at hand (for example, a neural network with at least 2-3 hidden layers), but the execution time would also increase as a result (perhaps more than we would like it to). Striking a balance between accuracy and computation time is always important, and in our particular case, the Logistic Regression model does just that: it provides decently accurate results while still minimizing the time it takes to be trained.