Classifying Review Sentiment with Bag-of-Words Features

Submitted By - Priyanshi Somani

Data -

The reviews used to classify sentiment in this report are collected from 3 domains - imdb.com, amazon.com, and yelp.com. In the training data, each review is a single sentence, and has a binary label indicating the sentiment of the review (1 for positive and 0 for negative) as shown in Figure 1(a) below. The test data has 200 single sentence reviews from amazon, yelp and imdb as shown in Figure 1(b) below. The task given is to classify the sentiment of the review. The training data is used to build and train the machine learning classifier.

Figure 1a

Figure 1b

Data Preprocessing -

Machine learning models do not understand words or sentences, instead each review needs to be converted into a vector where words have an associated feature value. This report uses a bag-of-words model for feature representation. Without any preprocessing of raw reviews, the size of the vocabulary is 4510. This consists of words like '00' , '1971', '700w', 'absolute', 'absolutel', 'absolutely', 'absolutley' etc. Hence, in order to decrease the size of the vocabulary, we need to preprocess the raw reviews. Below are the steps used to preprocess raw reviews -

a) Remove punctuation, non-English and non text characters
b) Make all words in reviews lowercase
c) Removing all stop words (Stop word corpus was taken from the NLTK), such as a and the,which are often used in text, but do not include specific information regarding sentiment.
d) Stemming words (i.e. bringing all different forms of a word to its root form). For example, {keeps, keeping, keep} will be reduced to keep.
e) Spelling Correction (used python package pyspellchecker for spelling correction)

Results after preprocessing the reviews are shown in the table below -

Raw Reviews	***Clean Reviews (after preprocessing)***
Basically the service was very bad.	basic service bad
Bad Choice.	bad choice
The only thing that disappoint me is the infra red port (irda).	thing disappoint infra red port ida
horrible, had to switch 3 times.	horrible switch time
It feels poorly constructed, the menus are difficult to navigate, and the buttons are so recessed that it is difficult to push them	feel poorly construct menu difficult navid button recess difficult push

After preprocessing, in order to create a vocabulary and convert raw reviews in feature vectors I have used a experimented with both a Count Vectorizer and TF-IDF vectorizer with the following settings -

a) minimum frequency of occurrence as 2, so that extremely rare words are not used to build the vocabulary
b) Used ngram range as [1,2] to include both one-word and two-word tokens. I used this primarily because some negative reviews have phrases like 'not good' to depict a negative sentime.

After preprocessing, the size of the vocabulary dropped to 2176. Hence every review is represented as a feature vector of size 2176.

Logistic Regression

The above generated feature matrix consisting of 2400 rows and 2176 columns is fed into a logistic regression classifier which outputs a binary label 1 or 0 corresponding to positive or negative sentiment respectively. In order to avoid overfitting, I performed 10-Fold cross validation and results are shown in Figure 2. From figure 2, we see that the training error rate is very low <5% however the error rate on test data is much higher ~20%. In order to control the model's complexity and its tendency to overfit, I

performed L2 regularization by ranging the value of hyperparameter c in the Logistic Regression model from 0.01 to 100. The results are shown in Figure 3


```
10-Fold Cross Validation
Average Training Error Rate: 0.048
Average Testing Error Rate: 0.199
```

Figure 2

Figure 2

We can see from Figure 3 how when the regularization strength is low, i.e. low penalty is applied for a complex model, the model fits perfectly on train data, yielding an error rate very close to 0%. In contrast, we see how the test error rate for such a model is much higher ~23%. This shows that this is an overfitted model. As we increase the penalty, although the training error rate increases, the error rate on test data decreases showing that the model is better fitting to unseen data. Around alpha = 1, we see the best model i.e. lowest error rate. And after that we see how the error rate on both train and test data starts to rise. This is because as we keep increasing the penalty, the model is unable to fit the data at all i.e. the model starts to underfit leading to a higher error rate.

Another hyperparameter I tuned was the solver used in the optimization problem. I compared the results for {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'} solvers. Below is the table showing the results of the same -

Solver	Train Error Rate	Test Error Rate
newton-cg	4.78%	19.88%
lbfgs	4.78%	19.88%
liblinear	4.76%	19.88%
sag	4.78%	19.92%
saga	4.78%	19.92%

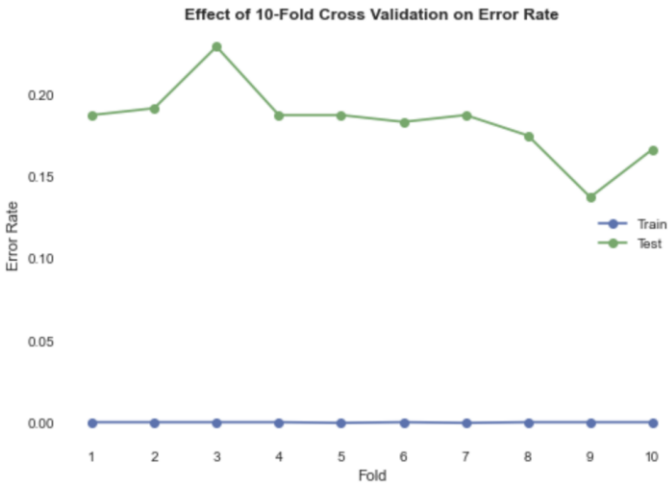
Neural Network

Architecture of Neural Network -

The neural network consists of a Sequential model with an Embedding Layer, Global Max Pooling 1D Layer, and two Dense Layers. The final dense layer has a sigmoid activation function so that we get output as probabilities between 0-1. The loss function used for optimization is binary-crossentropy as the task at hand is binary classification. Below is the summary of the model -

```
Model: "sequential_39"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_29 (Embedding)    (None, 100, 50)           184050

 global_max_pooling1d_29 (Gl  (None, 50)               0
 obalMaxPooling1D)

 dense_88 (Dense)            (None, 10)                510

 dense_89 (Dense)            (None, 1)                 11

=================================================================
Total params: 184,571
Trainable params: 184,571
Non-trainable params: 0
_________________________________________________________________
```

I used the Keras Tokenizer (this is synonymous to sklearn's Count Vectorizer) on clean reviews to generate feature vectors. In order to check if the model was overfitting on training data, I applied 10-Fold cross validation. The results of 10-fold cross validation on training and validation sets is shown in Figure 4.




```
10-Fold Cross-Validation Avg. Error Rate: 0.183 +/- 0.022
```

Figure 4

The above figure shows that the training error rate across all folds is almost close to 0. However, the error rate on test data is much higher (~18%). This shows that the model is overfitting on training data. Several parameters can be tuned before and after the model is tuned. Some of them are mentioned below -

i***. Giving different padding length to the feature vector after applying Keras Tokenizer*** - The reviews are of varied lengths, as shown in Figure 5 below. And hence, varying the length of padding as 10,50,100,200 significantly affects the cross validation accuracy. This is shown below. The lowest error rate (~17% with a confidence interval of 2.1%) is achieved when the feature vector is of size 50. We can see as we increase the size of the vector the error rate does not necessarily decrease as the model starts to overfit on the data. Furthermore, as we increase the size of the feature vector the training time also significantly increases and this is a tradeoff for using long feature representations.

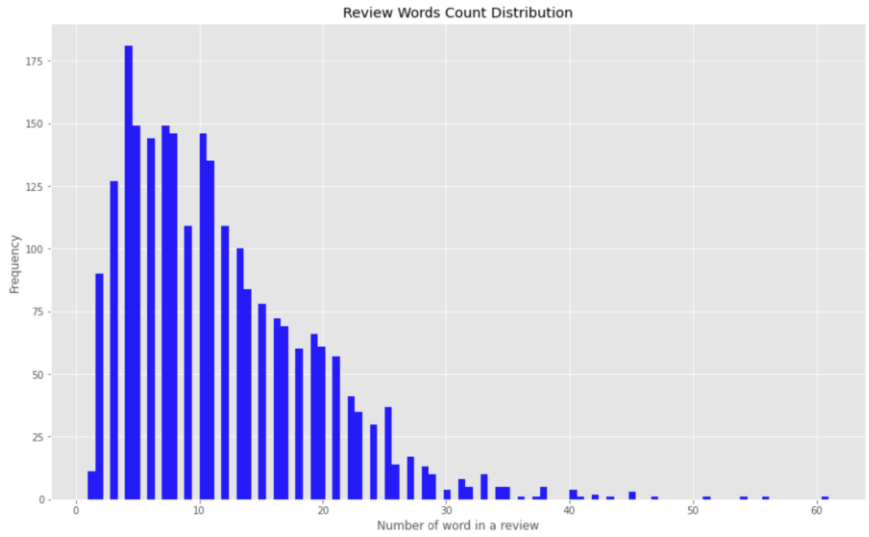


Figure 5

```
Pad Seq Len (10) 10-Fold Cross-Validation Avg. Error Rate: 0.220 +/- 0.024
Pad Seq Len (50) 10-Fold Cross-Validation Avg. Error Rate: 0.177 +/- 0.021
Pad Seq Len (100) 10-Fold Cross-Validation Avg. Error Rate: 0.184 +/- 0.021
Pad Seq Len (200) 10-Fold Cross-Validation Avg. Error Rate: 0.177 +/- 0.020
```

ii. Modifying hyper-parameters like batch size and number of epochs. The results of the same are shown in below -

```
No.of Epochs: 10
Batch Size (10) 10-Fold Cross-Validation Avg Error Rate: 0.184 +/- 0.019
Batch Size (20) 10-Fold Cross-Validation Avg Error Rate: 0.176 +/- 0.021
Batch Size (40) 10-Fold Cross-Validation Avg Error Rate: 0.172 +/- 0.021
Batch Size (60) 10-Fold Cross-Validation Avg Error Rate: 0.170 +/- 0.017
Batch Size (80) 10-Fold Cross-Validation Avg Error Rate: 0.168 +/- 0.016
Batch Size (100) 10-Fold Cross-Validation Avg Error Rate: 0.166 +/- 0.016
No.of Epochs: 50
Batch Size (10) 10-Fold Cross-Validation Avg Error Rate: 0.193 +/- 0.021
Batch Size (20) 10-Fold Cross-Validation Avg Error Rate: 0.188 +/- 0.017
Batch Size (40) 10-Fold Cross-Validation Avg Error Rate: 0.188 +/- 0.014
Batch Size (60) 10-Fold Cross-Validation Avg Error Rate: 0.181 +/- 0.026
Batch Size (80) 10-Fold Cross-Validation Avg Error Rate: 0.178 +/- 0.016
Batch Size (100) 10-Fold Cross-Validation Avg Error Rate: 0.183 +/- 0.018
No.of Epochs: 100
Batch Size (10) 10-Fold Cross-Validation Avg Error Rate: 0.200 +/- 0.020
Batch Size (20) 10-Fold Cross-Validation Avg Error Rate: 0.190 +/- 0.028
Batch Size (40) 10-Fold Cross-Validation Avg Error Rate: 0.186 +/- 0.026
Batch Size (60) 10-Fold Cross-Validation Avg Error Rate: 0.180 +/- 0.020
Batch Size (80) 10-Fold Cross-Validation Avg Error Rate: 0.183 +/- 0.019
Batch Size (100) 10-Fold Cross-Validation Avg Error Rate: 0.190 +/- 0.021
```

From the above results we can see that increasing the number of epochs does not always decrease the test error rate; however, increasing the batch size to the neural network model leads to an decrease in error rate in some cases. As we increase the number of observations in a batch to look at before making weight updates, the error rate decreases slightly.

iii. Varying Optimizer - Keras offers various algorithms for optimization, I tested 5 different optimizers and the results are shown below -

```
Optimizer (adam) Avg 10-Fold Cross-Validation Error Rate: 0.150 +/- 0.000
Optimizer (RMSprop) Avg 10-Fold Cross-Validation Error Rate: 0.163 +/- 0.000
Optimizer (sgd) Avg 10-Fold Cross-Validation Error Rate: 0.471 +/- 0.000
Optimizer (Adamax) Avg 10-Fold Cross-Validation Error Rate: 0.217 +/- 0.000
Optimizer (Ftrl) Avg 10-Fold Cross-Validation Error Rate: 0.546 +/- 0.000
```

From the results above we can see that adam, rmsprop and adamax optimizer perform much better than sgd and ftrl optimizers. These models generalize better for test reviews giving lower error rate.

iv. Varying the learning rate - I picked the adam optimizer, batch size of 100 and 50 epochs to conduct further analysis. Learning rate is a very important hyperparameter to tune in neural networks as it controls the weight update at the end of each batch. The result of varying learning rate on training and validation set is shown below -

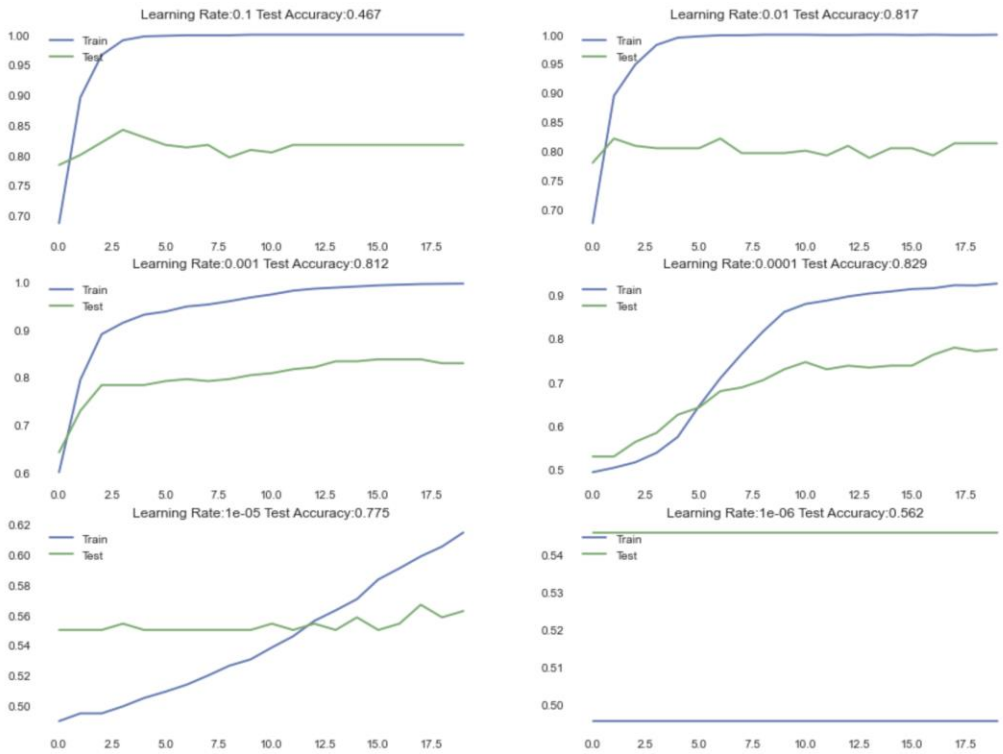


Figure 6

Decision Tree-

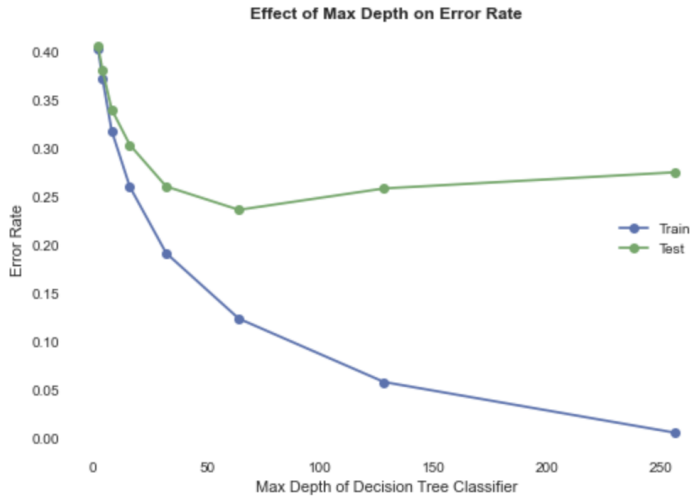
The third model I built was a decision tree classifier. I used a Count Vectorizer to transform reviews into feature vectors which were fed into a Decision tree classifier. The results of applying 10-fold cross validation are shown below.


```
10-Fold Cross-Validation Avg. Error Rate: 0.278 +/- 0.030
```

Figure 7

From the figure above, we can see the error rate on train data is almost close to 0 while the error rate on test data is much higher clearly indicating model overfitting. The max depth of this tree is 393. I

varied this hyperparameter on various values to find the ideal depth of the tree. This can prevent overfitting as the tree does not grow until all training samples are learnt, but stops at a fixed value. The results of limiting the depth of the decision tree are shown below -




```
Max Depth (64) 10-Fold Cross-Validation Avg. Error Rate: 0.236 +/- 0.023
```

Figure 8

By limiting the maximum depth of the tree to 50 or 70 leads to lower test error rate, showing that these trees generalize better on unseen data. Although complex trees give almost 0 error rate on train data, they do not perform well on unseen data because they are overfitted. The lowest error rate is achieved when the max depth of the tree is 64.

Hyperparameter tuning -

i. ***Modifying the criterion*** used by the decision tree classifier and the results are shown below -

```
Criterion (entropy) 10-Fold Cross-Validation Avg. Error Rate: 0.242 +/- 0.015
Criterion (gini) 10-Fold Cross-Validation Avg. Error Rate: 0.237 +/- 0.024
```

We can see from the results above that Gini criterion leads to lower error rate as it aims to minimize misclassification. Entropy also gives a comparable error rate, however slightly higher and also takes more time.

ii. ***Modifying the minimum number of samples in the leaf node*** for different values. This hyperparameter adjusts the minimum number required for a decision tree node to be a leaf. The default value is 1. The results for varying the minimum samples in leaf node are shown below -

```
Min Samples in Leaf (1) 10-Fold Cross-Validation Avg. Error Rate: 0.242 +/- 0.024
Min Samples in Leaf (2) 10-Fold Cross-Validation Avg. Error Rate: 0.245 +/- 0.024
Min Samples in Leaf (3) 10-Fold Cross-Validation Avg. Error Rate: 0.234 +/- 0.023
Min Samples in Leaf (4) 10-Fold Cross-Validation Avg. Error Rate: 0.250 +/- 0.019
Min Samples in Leaf (5) 10-Fold Cross-Validation Avg. Error Rate: 0.254 +/- 0.022
Min Samples in Leaf (10) 10-Fold Cross-Validation Avg. Error Rate: 0.282 +/- 0.028
Min Samples in Leaf (20) 10-Fold Cross-Validation Avg. Error Rate: 0.322 +/- 0.024
Min Samples in Leaf (50) 10-Fold Cross-Validation Avg. Error Rate: 0.383 +/- 0.022
Min Samples in Leaf (100) 10-Fold Cross-Validation Avg. Error Rate: 0.397 +/- 0.019
```

We can see that when the default value is used the error rate is around ~24%. As we increase the number of samples in leaf nodes to 3, the error rate drops. This shows that at the default value of this

parameter the model was slightly overfit. As we increase the number of samples in leaf nodes even further, the error rate increases. This is expected as the decision tree starts to underfit and is unable to classify positive and negative sentiment.

Misclassification -

The best classifier is chosen on the basis of lowest test error rate. The results of all the three models are shown in the table below.

Model	***Test Error Rate***	***Confidence***
Logistic Regression (with Count Vectorizer)	19.9%	+/- 4.8%
Sequential Neural Network (Keras Tokenizer)	16.6%	+/- 2.1%
Decision Tree Classifier (Count Vectorizer)	23.6%	+/- 2.3%

The Sequential Neural Network model achieves the lowest error rate. It is able to generalize better on unseen data and is least prone to overfitting.

The confusion matrix for the above model is shown below -

```
After 80-20 split of Training Data
Error Rate on Test Data:0.175
CONFUSION MATRIX
Predicted    0    1
True
0          191   46
1           38  205
```

From the matrix above, we can see that 94 reviews have been misclassified. The misclassified reviews are shown below -

i. Positive Reviews misclassified as negative:

```
highly recommended age although younger set probably appreciate subtle reference certainly appreciate one galley scene particular
advise look
def coming back bowl next time
place
convenient simple use get job done make car ride much smoother
someone shouldve invented sooner
performed
rare filmmaker take time tell worthy moral tale care love doesn't fall trap overly syrupy indulgent
recommend go anyone different brand cell player family
last night second time dining happy decided go back
get job done
happy complaint one regarding sound quality end
want healthy authentic ethic food try place
never forget
film sole bright spot jonah hill look almost unrecognizable fan recent superb due amount weight lost interim
sad movie good
battery working well replacement original came phone year ago
```

ii. Negative Reviews misclassified as positive:

```
dont go looking good food
new location need complete overhaul
result well shame
buck head really expect better food
zillion time away reality
kind flop around
easy watch
came back today since relocated still impressed
highly recommended
refuse refund replace
need least min get phone book time first turn  life short
absolutely warmth charm scene character
performance improved improvisation actor twice much worry whether there delivering line well whether line good
problem thought actor playing villain low rent michael inside
```

Generally the model makes a few mistakes -

- The model is unable to understand the meaning of certain words like ‘complaint’ ‘overhaul’ ‘awkward’ ‘infuriating’. These words clearly denote a negative sentiment, but since they do not occur often the model is not able to learn the sentiment associated with these words. Using a valence dictionary, where words are weighted on the basis of their polarity, can be a helpful input to the model.
- The model also makes mistakes in reviews where phrases like ‘dont go’ ‘not happy’ etc occur as the model tokenises reviews only on a word level and not n-gram level.
- The model also makes mistakes where the review in itself is difficult to classify. More contextual learning can help increase the accuracy of such a model.

Conclusion

Using the above-mentioned sequential model on test data gives the below results -

Error Rate on Leaderboard - **0.18667.** This value is slightly higher than the cross-validation error rate but falls within the confidence interval. I expected these results as the vocabulary is built using the training data, and generally the error rate on unseen data will be slightly higher.