MAY 26, 2024

PERFORMANCE ASSESSMENT

D213 TASK 2

ROBERT PATTON
WESTERN GOVERNORS' UNIVERSITY
Rpatt33@wgu.edu

Table of Contents

Part I: Research Question

A. Purpose of Data Analysis

1) Using sentiment analysis, can we predict positive and negative reviews from customers in the amazon data set?
2) The goal of this analysis is to build and train a model that can help us predict the sentiment value of customer reviews.
3) For this performance assessment, I used recurrent neural networking (RNN) for sentiment analysis. Because I am using sequential data, or data that is organized in a sequence where order is of great significance, RNN handles this type of data successfully (Kostadinov 2017). Having sequential data and identifying a need for a recurrent neural network, I was able to perform text classification that allowed me to train a model to make predictions on text sequences for amazon reviews.

Part II: Data Preparation

B. Summarize Exploratory Data Analysis

1) A. Upon exploration of the data, I examined the first 30 rows using the .head(30) function to look for any unusual characters. It was evident that special characters existed as I found exclamation marks, quotations, multiple periods in a row.

```
#Look at first 5 rows
df.head(15)
```

	Review	Sentiment
0	So there is no way for me to plug it in here in the US unless I go by a converter.	0
1	Good case, Excellent value.	1
2	Great for the jawbone.	1
3	Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!!	0
4	The mic is great.	1
5	I have to jiggle the plug to get it to line up right to get decent volume.	0
6	If you have several dozen or several hundred contacts, then imagine the fun of sending each of them one by one.	0
7	If you are Razr owner...you must have this!	1
8	Needless to say, I wasted my money.	0
9	What a waste of money and time!.	0
10	And the sound quality is great.	1
11	He was very impressed when going from the original battery to the extended battery.	1
12	If the two were seperated by a mere 5+ ft I started to notice excessive static and garbled sound from the headset.	0
13	Very good quality though	1
14	The design is very odd, as the ear "clip" is not very comfortable at all.	0

1) B. Vocabulary size was figured out by using the Tokenizer() function. The screen shot below shows the vocabulary size and the code used to determine such.

```
#Identify vocab size
tokenizer=Tokenizer()
tokenizer.fit_on_texts(df['Review'])
vocab_size=len(tokenizer.word_index)+1
print("Vocabulary size:", vocab_size)
```

```
Vocabulary size: 1905
```

1) C. Word embedding length was configured using the code in the screen shot below. We need to embed words as numbers because it allows machines to deal with words in that manner, where computers can use vectors assigned to words that were created in a high-dimensional space to make predictions (Madushanka 2023).

```
#Calculate embedding dimension
max_sequence_embedding= int(round(np.sqrt(np.sqrt(vocab_size)), 0))
max_sequence_embedding
```

```
7
```

1) D. The maximum sequence length is fairly straightforward and is simply the maximum length of the longest sentence in the data frame after we have finished data pre-processing. There for I started my max sequence length at 30, as I also had to pad the rest of the reviews to 30 characters long as well.

```
#Determine review lengths
review_length=[]
for char_len in reviews:
    review_length.append(len(char_len.split(" ")))

review_max=np.max(review_length)
review_min=np.min(review_length)
review_median=np.median(review_length)
print("Max length of sequences is: ", review_max)
print("Mid length of sequences is: ", review_median)
print("Min length of sequences is: ", review_min)
```

```
Max length of sequences is:  30
Mid length of sequences is:  9.0
Min length of sequences is:  1
```

2) Tokenization is similar to that of learning a new language, where instead of trying to jump in and read a full sentence of words you do not know, you begin with breaking it down word by words to understand roots and so forth. Tokenization does just that so that machines have an easier time digesting and understanding units of text (Awan 2023). The goal is to make it easier for algorithms to identify patterns by converting text into tokens without losing context, essentially it is the dissecting of a sentence for a machine to understand its anatomy (Awan 2023). The screen shot below shows an example of tokenization. You can see the sentence untokenized first, then broken down into segments of the sentence.

```
 'you can not answer calls with the unit, never worked once']
```

```
#Look for unusual characters and grammar
description_list=[]
stop_words=stopwords.words('english')
for description in df.Review:
#Remove special characters and punctuation
    description=re.sub("[^a-zA-Z]", " ", description)
#Perform tokenization
    description=nltk.word_tokenize(description)
#Implement lemmatization
    lemma=nltk.WordNetLemmatizer()
    decription= [lemma.lemmatize(word) for word in description]
print(description)
```

```
['you', 'can', 'not', 'answer', 'calls', 'with', 'the', 'unit', 'never', 'worked', 'once'
```

3) Similar to tokenization, padding adds extra tokens to a sequence to ensure that all sequences are the same length. Padding is necessary because models tend to operate best on a fixed-sized input data, padding addresses this issue so that shorter sequences are padded to the length of the max sequence in the dataset (eitca.org 2023). It is important to note that padding tokens do not carry any actual significance, other than properly structuring the data to allow a machine learning model to work with the fixed-sized data. For example, the max length of sequences in my data in 30, so if a sequence length contains only three words, we would need to pad this sequence with 27 padded tokens. For this task, the padded tokens are added at the end of the sequences by defining the padding_type as ‘post’. The screen shot below provides an example of the code for padding and an example of a padded sequence.

```
#Pad the split data
padding_type='post'
trunc_type='post'
#Train data
sequences_train= tokenizer.texts_to_sequences(X_train)
padded_train= pad_sequences(sequences_train, maxlen=review_max, padding=padding_type, truncating=trunc_type
#Test data
sequences_test= tokenizer.texts_to_sequences(X_test)
padded_test= pad_sequences(sequences_test, maxlen=review_max, padding=padding_type, truncating=trunc_type)
```

```
#Display padded sequence
import sys
np.set_printoptions(threshold=sys.maxsize)
padded_train[1]
```

```
array([590,  50,  15, 610,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0], dtype=int32)
```

4) The data provided by WGU came organized into sentiment values of 0/1. The negative reviews are associated with the value 0 and the positive ones with the value of 1. Other options exist as well, that is, if I were to decide that some reviews were neutral, I could create a 0,1,2 sentiment list. For the model, the activation function used was sigmoid because the sentiment data is binary. I also used the relu activation function of some layers of the model, relu is great because it is usually used in the hidden layers of a neural network and adds non-linearity (Saxena 2024).

```
#Create model

#Define early stopping
early_stopping_monitor = EarlyStopping(patience=2)

model = Sequential()
model.add(Embedding(vocab_size, max_sequence_embedding, input_shape = (training_padded.shape[1],)))
model.add(GlobalAveragePooling1D())
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(2, activation='sigmoid'))
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam', metrics=['accuracy'])
model.summary()
```

5) The steps to prepare the data for analysis is as follows:
    a. Read in the data set using pandas.
    b. Examine the data using the .head(), shape, and value_counts() functions.
    c. Check for unusual and abnormal characters in the reviews column.
    d. Clean the data and get rid of any duplicate values and manage any missing values.
    e. Lower case the reviews.
    f. Remove punctuation in the reviews.
    g. Remove insignificant stop words.
    h. Implement tokenization to improve machine learning of reviews for prediction.
    i. Build a two dimensional array using NumPy for sentiment ratings.
    j. Split the data into an 80/20 split.
    k. Apply fit_on_texts tokenizer method to the training set.
    l. Implement post-padding to sequences.
    m. Run a model using the Sequential() package and fit the final model.
    n. Convert the padded and final training and test sets to a NumPy array.
6) The prepared data, including the cleaned data, is uploaded with the task two submission.

Part III: Network Architecture

C. Network Used

1) I used the model.summary() function to retrieve a summary of the keras model outputs. This output gives us an explanation of the layer names, type, shape, and the number of trainable parameters. A screen shot of these outputs is shown below.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 30, 7)	13,335
global_average_pooling1d (GlobalAveragePooling1D)	(None, 7)	0
dense (Dense)	(None, 100)	800
dense_1 (Dense)	(None, 50)	5,050
dense_2 (Dense)	(None, 2)	102

Total params: 19,287 (75.34 KB)
Trainable params: 19,287 (75.34 KB)
Non-trainable params: 0 (0.00 B)

2) The five layers shown in the above model output can be explained as follows, there are a total of 19,287 parameters, with all 19,287 being trainable:
    a) The first layer is the embedding layer, which is an input layer type with three output dimensions and 13,335 parameters. The output shape has no specific number of batch size to be used, 30 number of words in each review, and each word is represented in 7 vector layers.
    b) The second layer is a flatten layer using Global_Average_Pooling1D(). It has two dimensions, again no specific number of batch size and 7 vector layers. This layer has 0 parameters.
    c) The third layer is a dense layer, which is a hidden layer with two output dimensions. It is set to use 100 neurons using a ‘relu’ activation function, which was explained in part 2, B4. This layer has 800 parameters.
    d) The fourth layer is also a dense layer, another hidden layer with two output dimensions. The only difference is that this layer has been set to 50 neurons using ‘relu’ activation function. This layer has 5,050 parameters.

e) The final layer is a hidden dense layer as well. It has two neuron layers, but the activation function has been set to ‘sigmoid’ because of the binary output for the sentiment column. This layer has 102 parameters.

3) Hyperparameters chosen for the model are justified as such:
   a) The activation functions I used were ‘relu’ and ‘sigmoid’. RELU is the most popular activation function for deep learning models as it has a linear behavior which makes a neural network easier to optimize (Krishnamurthy & Whitfield 2024). RELU was used for the first and second hidden layers. For the final layer I used the sigmoid function because it is good for binary classification, as my sentiment values are 0 or 1.
   b) In determining how many nodes to use, it generally comes down to a trial and error method. Adding layers to a deep learning model helps to reduce overfitting and underfitting, leading to an increased accuracy of the model. When determining the best option for node values, any analyst could start at 50 and add more nodes or more hidden layers with different node lengths to help the model perform more effectively. In my model, I began with 100 nodes for the third layer, then dropping it to 50 nodes in the fourth layer, and finally 2 in the final layer. It may not have been totally necessary to utilize that many nodes, but the model produced very good results.
   c) In choosing a loss function, I decided to use sparse_categorical_crossentropy. Loss functions help us determine how efficient a neural network models our training data, comparing the target and predicted output, where the end goal is to minimize loss between these outputs (Yathish 2022). Sparse categorical cross entropy is useful when there are two or more label classes, when labels are in integer form like that of my sentiment values of 0/1.
   d) The Adam optimizer is the most commonly used in the data science field, as it is known to be a very effective and the fastest optimizer for deep learning models. Therefore, being a sort of industry standard and go to for most analysts, I decided to use it for my model.
   e) The stopping criteria used for my model consisted of the EarlyStopping() function with a patience set to 2. Early stopping is a unique function that helps prevent overfitting of your model, where it intentionally stops the training process at a certain number of epochs before overfitting occurs (Pramoditha 2022).

f) The evaluation metric I chose to use was the accuracy metric. For the test set, my model had an accuracy score of 80%. For the training set, the model had an accuracy score of 98%. Overall, this tells me that my model is correctly predicting the sentiment of reviews very well.

Part IV: Model Evaluation

D. Model Training Process and Outcomes

1) As briefly mentioned above, early stopping criteria can be very helpful when determining how far a model will train before validation scores diminish. When running a model, an analyst can set a predetermined number of epochs for the model, however. How does one know what the ideal number of epochs should be? When utilizing the early stopping function, the model will run until it determines that validation is getting any better. We use a patience argument within the early stopping parameter as well, this allows us to tell the model how much further beyond the last best epoch to stop the model, usually 2 to 3 patience levels. An example of the final training epoch can be seen below.

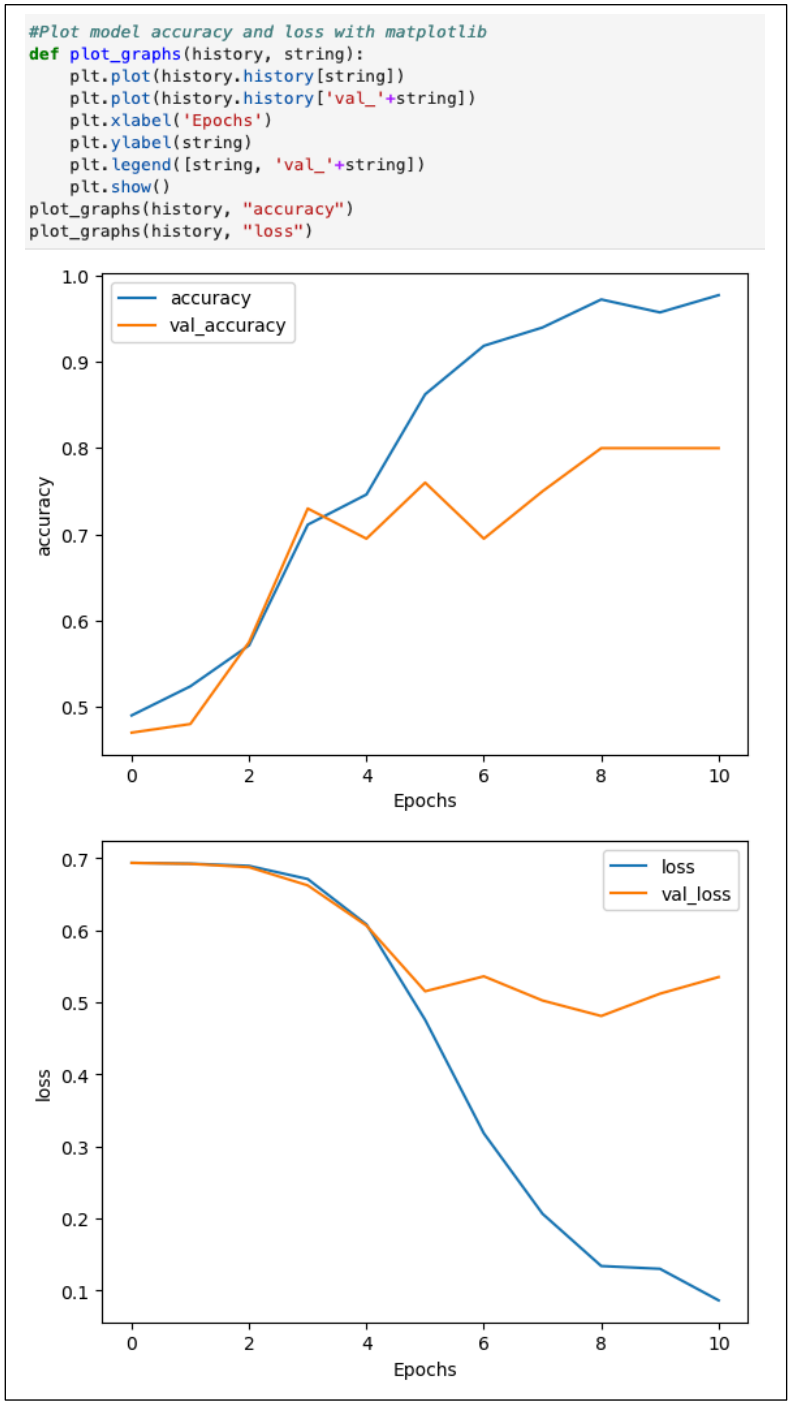
```
#Fit the model
num_epochs=15
history= model.fit(training_padded, training_label, batch_size=32, epochs=num_epochs, validation_data=(test_padded, test_label), callbacks=[early_stopping_monitor], verbose=True)
```

```
Epoch 1/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.5048 - loss: 0.6936 - val_accuracy: 0.4700 - val_loss: 0.6932
Epoch 2/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5626 - loss: 0.6924 - val_accuracy: 0.4700 - val_loss: 0.6918
Epoch 3/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5324 - loss: 0.6876 - val_accuracy: 0.4750 - val_loss: 0.6862
Epoch 4/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5935 - loss: 0.6702 - val_accuracy: 0.5650 - val_loss: 0.6598
Epoch 5/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8504 - loss: 0.6090 - val_accuracy: 0.8050 - val_loss: 0.5775
Epoch 6/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8483 - loss: 0.4825 - val_accuracy: 0.7550 - val_loss: 0.5029
Epoch 7/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9251 - loss: 0.3148 - val_accuracy: 0.8000 - val_loss: 0.4578
Epoch 8/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9434 - loss: 0.2188 - val_accuracy: 0.8000 - val_loss: 0.4594
Epoch 9/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9662 - loss: 0.1455 - val_accuracy: 0.7900 - val_loss: 0.5181
```

2) Adjusting for overfitting or underfitting can be done with a few different adjustments to the model. We can reduce the number of epochs, add more layers to the model, or add more nodes to the layers in the model. It can be a little bit of a trial and error task, as I found out myself, and an analyst may to fine tune such parameters until the model results improve. For example, I had to originally started out with 20 epochs and after trying 5, and then 10, I found that 15 was the sweet spot for my model.

3)

```
#Plot model accuracy and loss with matplotlib
def plot_graphs(history, string):
    plt.plot(history.history[string])
    plt.plot(history.history['val_'+string])
    plt.xlabel('Epochs')
    plt.ylabel(string)
    plt.legend([string, 'val_'+string])
    plt.show()
plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```



4) My model prediction accuracy can be seen below. I used the .evaluate() function on my test data to generate model accuracy and loss. The model accuracy came in at 80% with a loss of 53%. In addition, I verified my sentiment prediction is accurate with the following code seen below, showing that sentiment prediction is indeed accurate.

```
#Verify model accuracy on test data
score= model.evaluate(test_padded, test_label, verbose=0)
print(f'Test loss: {score[0]}/ Test accuracy: {score[1]}')
```

```
Test loss: 0.5349713563919067/ Test accuracy: 0.800000011920929
```

```
##Verfy prediction sentiment
i=42

print("Predicted review test:", X_test[i], "\n:")
print("Predicted:", "Negative" if predictions [i][0] >= 0.5 else "Positive", "review")
print("Actual:", "Negative" if y_test[i] == 0 else "Positive", "review")
```

```
Predicted review test: nice headset priced right
:
Predicted: Positive review
Actual: Positive review
```

Part V: Summary and Recommendations

E. The code used to save the trained network was model.save('my_model.keras').

F. Overall, using TensorFlow for this sentiment analysis project worked out very well. The model accuracy came in at 80%, which I feel is quite accurate at predict a customer's review as either positive or negative. However, because this data set only contained 1000 reviews, having a more robust data set would increase historical reviews that could improve the prediction accuracy. In addition, the vocabulary size was not as large, have a larger data set would increase the vocabulary size and that could improve model efficiency as well. Tokenization is an important tool in deep learning models such as sentiment analysis, performing this task improves the machines capability of recognizing words and being able to recognize various units of text. Using trial and error during model building impacted the outcome of the final model. By hyper tuning parameters, such as epoch size, node size, activation functions, and loss functions for example, I was able to produce a final model with an acceptable degree of accuracy.

G. Again, I believe a larger data set with more amazon reviews would improve the prediction accuracy for sentiment analysis. Therefore I would suggest pulling more data from review archives in an attempt to increase predication accuracy for positive or negative reviews.

Part VI: Reporting

H. The interactive development environment used for the task was Jupyter notebook and the file is uploaded with the task submission.

I. Sources

Kostadinov, S. (2019, November 10). *How recurrent neural networks work*. Medium. https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

Madushanka, T. (2023, February 1). *Word embedding: Understanding the fundamentals*. Medium. https://medium.com/zero-to/word-embedding-understanding-the-fundamentals-746748000700

EITCA Academy. (2023, August 5). *What is the function of padding in processing sequences of tokens?* https://eitca.org/artificial-intelligence/eitc-ai-tff-tensorflow-fundamentals/natural-language-processing-with-tensorflow/sequencing-turning-sentences-into-data/examination-review-sequencing-turning-sentences-into-data/what-is-the-function-of-padding-in-processing-sequences-of-tokens/#:~:text=By%20doing%20so%2C%20padding%20ensures,%22Deep%20learning%20is%20fascinating%22

Saxena, S. (2024, May 24). *Introduction to softmax for neural network*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/#:~:text=contains%20a%20dog.-,When%20to%20use%20Softmax%20vs%20ReLU,network%20to%20add%20non%2Dlinearity.

Krishnamurthy, B., & Whitfield, B. (2024, February 26). *An introduction to the ReLU activation function*. Built In. https://builtin.com/machine-learning/relu-activation-function

Yathish, V. (2022, August 4). *Loss functions and their use in neural networks*. Medium. https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9#:~:text=Loss%20Functions%20Overview,of%20J%20(average%20loss).&text=We%20can%20think%20of%20this,to%20minimize%20the%20net%20distance

Pramoditha, R. (2022, July 19). *Using early stopping to reduce overfitting in neural networks*. Medium. https://medium.com/data-science-365/using-early-stopping-to-reduce-overfitting-in-neural-networks-7f58180caf5b