Advanced Data Analytics

PA2 – Sentiment Analysis

Using Neural Networks

Jason Willis

College of Information Technology,

Western Governors University

Dr. Festus Elleh

Aug 24, 2022

Table of Contents for Each Rubric

Data Driven Decisions

The models below will delve into machine learning and artificial intelligence. In this performance assessment, we explore the use of neural networks and natural language processing (NLP) by using review and sentiment data from the UCI Sentiment Labeled Sentences Data sets. (Dua D., et. al., 2019)

A1 – Research Question

Using historical product reviews, what effect will Natural Language Processing and Neural Networks have on the ability to accurately predict future consumer sentiment?

A2 – Objectives and Goals

Using a combination of three sources of reviews, we are trying to use natural language processing pattern recognition to help understand and potentially predict consumer sentiment. By leveraging our data-driven recommendation model to predict customer sentiment, we can make more insightful suggestions to stakeholders.

A3 – Prescribed Network

Text classification and customer sentiment identification is performed by fitting a deep learning neural network and natural language processing models to the review data. Deep learning neural networks are a type of recurrent neural network which works well on sequential data. By leveraging the artificial neural network's input layer, multiple hidden layers and one output layer, the model was able to discern potential future reviews with the correct sentiment.

B1 – Data Exploration

Exploratory data analysis was performed on the data set. Here are a few explanations of what was performed:

- Presence of unusual characters (e.g., emojis, non-English characters, etc.) see Figure 1.
- Vocabulary size is the number of unique words in the data set (see Figure 2) once stop words and unusual characters are removed. (Misheva)
- Maximum word embedding length is a positional measurement of similar meaning to predict which words occur together, see Figure 3. It's the square root of the square root of your vocabulary size "also known as the 4th root", F. Elleh, 2022.
- Maximum length was chosen to help preserve the integrity of input data for accurate conclusions. Inputs that are shorter will be padded. After testing various sequence lengths, 200 was the chosen maximum sequence length since it provided the highest accuracy, see Figure 4.

```
# Character Count
commentary = df['review'].str.lower()
list_of_chars = []
for comment in commentary:
    for character in comment:
        if character not in list_of_chars:
            list_of_chars.append(character)
num_of_chars = len(list_of_chars)
print("Number of Characters: ", num_of_chars)
print(list_of_chars) # Notice the presence of unusual characters (B1a)
```

```
Number of Characters:  56
['s', 'o', ' ', 't', 'h', 'e', 'r', 'i', 'n', 'w', 'a', 'y', 'f', 'm', 'p', 'l', 'u', 'g', 'b', 'c', 'v', 'd', 'x',
'j', '4', '5', 'z', 'q', '+', 'k', '/', '7', '3', '6', '8', '0', '2', '1', ')', '(', '&', '$', '*', '%', '9', '#',
'[', ']', '\x96', '\t', '\n', 'é', '\x85', 'â', '\x97', 'ë']
```

Figure 1 - Presence of Unusual Characters

```
# Vocab Size (B1b)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(review_df['review']) # Fit tokenizer on reviews
print('Vocab Size: ', len(tokenizer.word_index)+1) # Vocab size is the number of unique words + 1 for the 0 index
```

```
Vocab Size:  5383
```

Figure 2 - Vocabulary Size

```
max_review_length = len(description)
print("Max Review Length: ", max_review_length)
```

```
Max Review Length:  72
```

Figure 3 - Max Character Length of Review

```
history = model.fit(training_padded, training_label, batch_size=32 ,epochs=num_epochs, validation_split = 0.3, callback
```

```
Epoch 1/20
39/39 [==============================] - 0s 5ms/step - loss: 0.6943 - accuracy: 0.4878 - val_loss: 0.6932 - val_accur
acy: 0.4905
Epoch 2/20
39/39 [==============================] - 0s 3ms/step - loss: 0.6927 - accuracy: 0.5171 - val_loss: 0.6926 - val_accur
acy: 0.5114
Epoch 3/20
39/39 [==============================] - 0s 3ms/step - loss: 0.6913 - accuracy: 0.5650 - val_loss: 0.6904 - val_accur
acy: 0.5152
Epoch 4/20
39/39 [==============================] - 0s 3ms/step - loss: 0.6814 - accuracy: 0.6195 - val_loss: 0.6841 - val_accur
acy: 0.4981
Epoch 5/20
39/39 [==============================] - 0s 3ms/step - loss: 0.6172 - accuracy: 0.7528 - val_loss: 0.6103 - val_accur
acy: 0.7557
Epoch 6/20
39/39 [==============================] - 0s 3ms/step - loss: 0.4441 - accuracy: 0.8528 - val_loss: 0.5095 - val_accur
acy: 0.7727
Epoch 7/20
39/39 [==============================] - 0s 3ms/step - loss: 0.2476 - accuracy: 0.9350 - val_loss: 0.4670 - val_accur
acy: 0.7879
Epoch 8/20
39/39 [==============================] - 0s 3ms/step - loss: 0.1558 - accuracy: 0.9593 - val_loss: 0.5180 - val_accur
acy: 0.7670
Epoch 9/20
39/39 [==============================] - 0s 3ms/step - loss: 0.1148 - accuracy: 0.9683 - val_loss: 0.5094 - val_accur
acy: 0.7803
```

Figure 4 - Model Accuracy

B2 – Tokenization

Tokenization is a separation process which splits strings into smaller amounts called tokens. These smaller bits can be as small as a character, large as a word or somewhere in-between. An index value is assigned to each "chunk" to help with training.

B3 – Padding Process

To improve performance, "padding" is a neural network technique is used by preserving a consistent shape of tensor dimensions. A neural network requires input to be of the same shape and size, so smaller input lengths are padded to keep the shape constant. After viewing Figure 5, you can see the padding happens prior to token 27, which equals "so".

```
# Reviews Encoded and Padded
print(reviews[0])
print(encoded_docs[0])
```

```
so there is no way for me to plug it in here in the us unless i go by a converter
[27, 49, 5, 60, 118, 12, 73, 6, 368, 7, 11, 67, 11, 1, 184, 640, 3, 77, 62, 4, 2215]
```

```
# view padded sequence for above example
print(padded_sequence[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    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    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    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    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    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    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0   27   49    5
   60  118   12   73    6  368    7   11   67   11    1  184  640    3
   77   62    4 2215]
```

Figure 5 - Padding Example

B4 – Categories of Sentiment

The data labels are set to two categories: 0 and 1; which equals zero for negative reviews and one for positive reviews. Please see Figure 6 for more detail.

```
# Tally Review Field Counts
print("Percentage of Pos/Neg Reviews:\n", df['label'].value_counts() / len(df['label']))
print("Totals:\n",df['label'].value_counts()) # Notice pos/neg ~approx equal
```

```
Percentage of Pos/Neg Reviews:
 1    0.504367
0    0.495633
Name: label, dtype: float64
Totals:
 1    1386
0    1362
Name: label, dtype: int64
```

```
# View Data Distro
sns.countplot(data=df, x='label', palette='viridis').set(title='Sentiment Data Distribution');
```

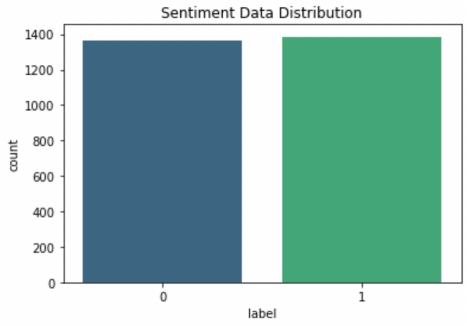


Figure 6 - Sentiment Categories

B5 – Steps to Prepare the Data

To prepare the data for analysis, the following steps were performed in accordance with Dr. Elleh's presentation (2022):

- Three data sources were imported and read into the notebook
- Data was checked for unusual characters

- Exploratory data analysis was performed to understand the dataset deeper, such as shape, identifying null values (if any), and I chose to convert the day field to a date field with an index.
- A NumPy array of ratings was created by encoding the data.
- Data is split into a training and testing subsets using approximately an 80/20 split.
    - Tokens were created on the training data set using the fit_on_texts() method.
    - A word index was also created on the training set.
- Pre-padding was used on the sequences to reach maximum lengths.
- The final model was fit; train/test sets were converted into arrays using NumPy.

B6 – Prepared Dataset

The submission includes prepared and cleaned data set: df_cleaned_stationary.csv

```
# Export padded train/test data
pd.DataFrame(X_train).to_csv('./Output/X_training_df.csv')
pd.DataFrame(X_test).to_csv('./Output/X_testing_df.csv')
pd.DataFrame(y_train).to_csv('./Output/y_training_df.csv')
pd.DataFrame(y_test).to_csv('./Output/y_testing_df.csv')
pd.DataFrame(X_validation).to_csv('./Output/X_validation_df.csv')
pd.DataFrame(y_validation).to_csv('./Output/y_validation_df.csv')
```

Figure 7 - Export Prepared Datasets

C1 – Model Summary

By using the Keras model.summary method (Figure 8), a summary is created of our model to help visualize the output of layer names, type, shape and the number of trainable parameters.

```
# B4 - Sentiment Analysis

activation = 'sigmoid' #'softmax'
loss = 'binary_crossentropy'  #'categorical_crossentropy'
optimzer = 'adam' #rmsprop

num_epochs = 20

# Define early_stopping_monitor
early_stopping_monitor=EarlyStopping(patience=2)

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Dropout(.2, input_shape=(2,)),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(50, activation='relu'),
    tf.keras.layers.Dense(1, activation=activation)
])

model.compile(loss=loss, optimizer=optimzer, metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 72, 32)            115264

 dropout (Dropout)           (None, 72, 32)            0

 global_average_pooling1d (G  (None, 32)               0
 lobalAveragePooling1D)

 dense (Dense)               (None, 100)               3300

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 1)                 51

=================================================================
Total params: 123,665
Trainable params: 123,665
Non-trainable params: 0
_________________________________________________________________
```

Figure 8 - Keras Model Summary

C2 – Network Architecture

By using the Keras model.summary method (Figure 8), you can decipher the following:

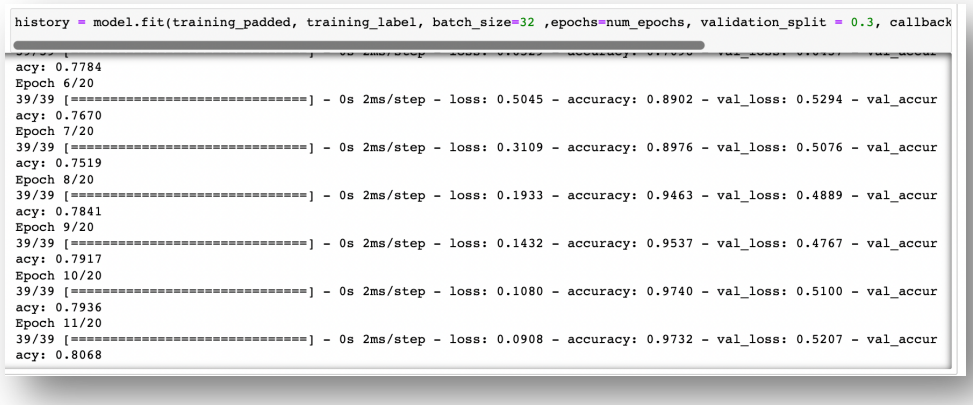
1. Embedding Layer – input of 115,246 parameters
2. Dropout layer – zero parameters

3. Flatten – zero parameters
4. Dense Layer – input of 3,300 parameters
5. Dense Layer – input of 5,050 parameters
6. Dense Layer – input of 50 parameters

The total trainable parameters are 123,665.

C3 – Hyperparameters

Our classification problem was binary, so the activation function chosen was the Sigmoid function to help predict between 0 and 1 in the label series. The number of nodes was determined by experimentation relevant to the model's accuracy. Since our classification model was binary, the loss function chosen was also binary, e.g. binary crossentropy, to help with classification. The "adam" optimizer was used in the lectures and worked well, so I continued using it since it has a high level of adaptability and able to handle input noise, which I thought could be present in three different review sources. Stopping criteria as also utilized to help stop the neural network prior to overfitting by stopping the training (with a patience of 2) when the validation score ceases to improve. I chose a higher number of epochs of 20 to use hoping between a higher number and the stopping criteria, a more optimized number of runs might emerge. Looking at both Figure 9 and 10, we can see the training accuracy (97%), validation accuracy (73%) and testing accuracy (72%).



```
history = model.fit(training_padded, training_label, batch_size=32 ,epochs=num_epochs, validation_split = 0.3, callback
acy: 0.7784
Epoch 6/20
39/39 [==============================] - 0s 2ms/step - loss: 0.5045 - accuracy: 0.8902 - val_loss: 0.5294 - val_accur
acy: 0.7670
Epoch 7/20
39/39 [==============================] - 0s 2ms/step - loss: 0.3109 - accuracy: 0.8976 - val_loss: 0.5076 - val_accur
acy: 0.7519
Epoch 8/20
39/39 [==============================] - 0s 2ms/step - loss: 0.1933 - accuracy: 0.9463 - val_loss: 0.4889 - val_accur
acy: 0.7841
Epoch 9/20
39/39 [==============================] - 0s 2ms/step - loss: 0.1432 - accuracy: 0.9537 - val_loss: 0.4767 - val_accur
acy: 0.7917
Epoch 10/20
39/39 [==============================] - 0s 2ms/step - loss: 0.1080 - accuracy: 0.9740 - val_loss: 0.5100 - val_accur
acy: 0.7936
Epoch 11/20
39/39 [==============================] - 0s 2ms/step - loss: 0.0908 - accuracy: 0.9732 - val_loss: 0.5207 - val_accur
acy: 0.8068
```

Figure 9 - Model Fit

```
# 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: 0.5053887367248535 / Test Accuracy: 0.7654545307159424
```

Figure 10 - Evaluation Metric

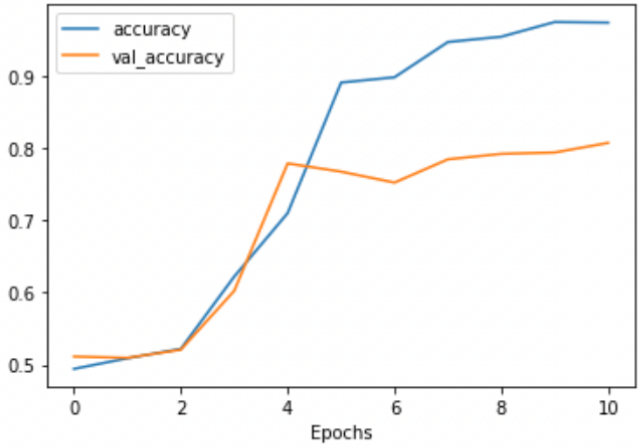
D1 – Stopping Criteria

As mentioned in C3 – Hyperparameters, I chose a higher number of epochs of 20 to use strategizing that between this higher epoch number and the stopping criteria, a more optimized number of runs might emerge. I chose a patience of 2 on the early stopping monitor. The balance between the number of epochs and the stop criteria stopping model fitting once the accuracy stops improving provides an elevated confidence.

D2 – Training Process

In the model, we set epoch to 20 with an early stopping monitor and patience of 2. As we see in Figure 11, the validation scores continue to improve until training epoch 9 and then the following 2 fail to improve higher.

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.savefig('./Output/AccuracyPlot.png')
plt.show()
```




```
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epochs')
plt.legend()
plt.savefig('./Output/LossPlot.png')
plt.show()
```

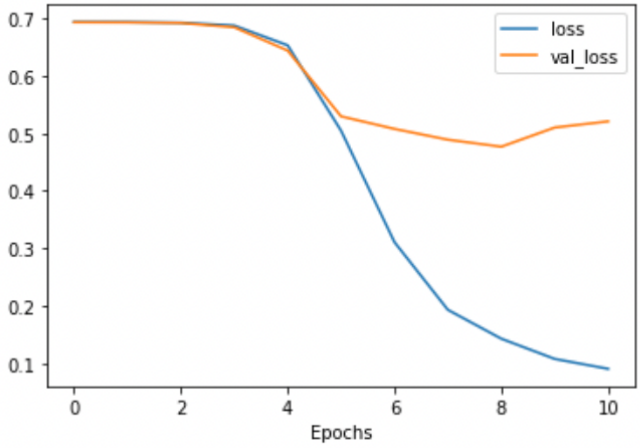


Figure 11 - Training Process Visualization

D3 – Fit

Overfitting is when the model learns the training dataset too well and doesn't adapt to testing data. In order to help avoid this situation, the keras.layers.Dropout method was used. (Figure 8)

D4 – Predictive Accuracy

The submission discusses the predictive accuracy of the trained network using the selected evaluation metric from part D2.

As seen in Figure 10, the model prediction accuracy on test data is 76.5%, which isn't too bad. A testing model was provided (Figure 11) to provide a couple sentences not yet seen by the model. The model was able to predict these two sentences accurately.

Test Model

```
def predict_sentiment(review):
    tw = tokenizer.texts_to_sequences([review])
    tw = pad_sequences(tw, maxlen=72)
    prediction = int(model.predict(tw).round().item())
    if prediction == 0:
        return "Negative"
    elif prediction == 1:
        return "Positive"
    else:
        return "Houston, we have a problem"
    print("Prediction Label: ", sentiment_label[1][prediction])
```

```
test_sentence_one = "I enjoyed my journey with this class, I hope to leverage NLP skills at work soon!"
print("Sentence One Prediction: ", predict_sentiment(test_sentence_one))
test_sentence_two = "I hated this class, I'll never use NLP and hope to never take touch this subject again!"
print("Sentence Two Prediction: ", predict_sentiment(test_sentence_two))
```

```
Sentence One Prediction:  Positive
Sentence Two Prediction:  Negative
```

Figure 12 - Predictive Testing

E – Code

Provided in the zipped folder is a Jupyter notebook (D213-AdvancedDataAnalyticsPA2.ipynb) a hierarchical data formatted file (JWillis_D213_PA2_SentimentAnalysisModel.h5) and pdf export (JWillis_D213-AdvancedDataAnalytics_PA2.pdf) detailing the code used to support the implementation of the time series model. Additionally, there are the cleaned *.csv files and images of visualizations.

F – Functionality

Close to 3,000 reviews from Amazon, Yelp and the Internet Movie Database were collected and combined, cleaned, chunked into tokenized parts and padded into equal sized segments. From there, 123,665 parameters were split into test (20%) and training (80%) data sets to be processed by a natural language algorithm. The model leveraged the sentiments from customers reviews to train itself and record labels accordingly. Through many iterative training and tuning sessions, the model has been modified to predict new customers reviews and identify sentiments as positive or negative reviews.

G – Recommendations

Based on initial results, I recommend retrieving more training data to increase the test accuracy when analyzing testing data. With minimal test set thus far combined with attention to output details and adjusting parameter tuning accordingly, we have achieved an acceptable degree of 76.5% accuracy thus far. In addition, I would also add a more stratified dataset rather than binary input to increase exposure to realistic rating systems.

H – Reporting

Provided in this document is an accurately created both VS-Code and Jupyter Notebooks. Both a Jupyter notebook (D213-AdvancedDataAnalyticsPA2.ipynb) and pdf export (D213-AdvancedDataAnalyticsPA2.pdf has been provided in the zipped package.

I – Sources for Third-Party Code

- Help using Markdown: https://www.markdownguide.org/basic-syntax/
- MacTeX: https://tug.org/mactex/mactex-download.html
- Matplotlib Help: https://matplotlib.org/2.1.2/api/_as_gen/matplotlib.pyplot.plot.html
- Numpy Help: https://numpy.org/doc/stable/
- Pandas Help: https://pandas.pydata.org/docs/user_guide/index.html#user-guide
- Python Help: https://docs.python.org/3.9/library/index.html
- Scipy.stats Help: https://docs.scipy.org/doc/scipy/reference/tutorial/stats.html

J – Sources

See the references below:

References

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. Found Here: https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

Elleh, F. (2022). Lecture: D213 T2 – Welcome to D213 Advanced Data Analytics. Western Governors University. Found Here: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=cedbd86a-2543-4d9d-9b0e-aec4011a606d

Misheva, V., et. al. (n.d.) Sentiment Analysis in Python. DataCamp. Found Here: https://campus.datacamp.com/courses/sentiment-analysis-in-python/sentiment-analysis-nuts-and-bolts?ex=1