Creating a Recurrent Neural Network (RNN) to predict customer sentiment based on text based review

Michael Phipps, March 2023

Data is a combination of Amazon, yelp and imdb reviews from UC Irvine - ML Repository

In [19]:
```
import numpy as np
import pandas as pd
import sklearn
import keras
import re #regular expression
from sklearn import preprocessing
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix #, ConfusionMatrixDisplay
import nltk #natural language tool kit
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import os #can see current working dir- change with chdir
import datetime
import tensorflow as tf
#from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, SpatialDropou
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
nltk.download('omw-1.4')
nltk.download ('stopwords')
nltk.download ('punkt')
nltk.download ('wordnet')
os.chdir("C:\\Users\\mikep\\Documents\\WGU\\D213 Advanced Data Analytics\\Task 2")
os.getcwd()
```

```
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\mikep\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\mikep\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\mikep\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\mikep\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[19]:
```
'C:\\Users\\mikep\\Documents\\WGU\\D213 Advanced Data Analytics\\Task 2'
```

Import the data, join the 3 datasets and clean/prepare the data

In [2]:
```
# read the data from three text files
amazon_data = pd.read_csv('amazon_cells_labelled.txt', delimiter='\t', header=None)
imdb_data = pd.read_csv('imdb_labelled.txt', delimiter='\t', header=None)
yelp_data = pd.read_csv('yelp_labelled.txt', delimiter='\t', header=None)

# concatenate into one dataframe
df_a = pd.concat([amazon_data, imdb_data, yelp_data])

# set column names
df_a.columns = ["review", "sentiment"]

# convert reviews to lowercase
df_a['review'] = df_a['review'].str.lower()
```

Data Preparation. EDA and Data Cleaning

In [3]:
```
df_a.dropna()
df_a.shape
```

Out[3]:
```
(2748, 2)
```

In [4]:
```
#looking at chars in dataset
charlookup = df_a['review']
listchars = []
for review in charlookup:
    for char in review:
        if char not in listchars:
            listchars.append(char)
print(listchars)
```

```
['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', 'å', '\x97', 'ê']
```

In [5]:
```
df_a["sentiment"].value_counts()
```

Out[5]:
```
1    1386
0    1362
Name: sentiment, dtype: int64
```

Removing special characters, conjunctions, stopwords and performing lemmatization & tokenization

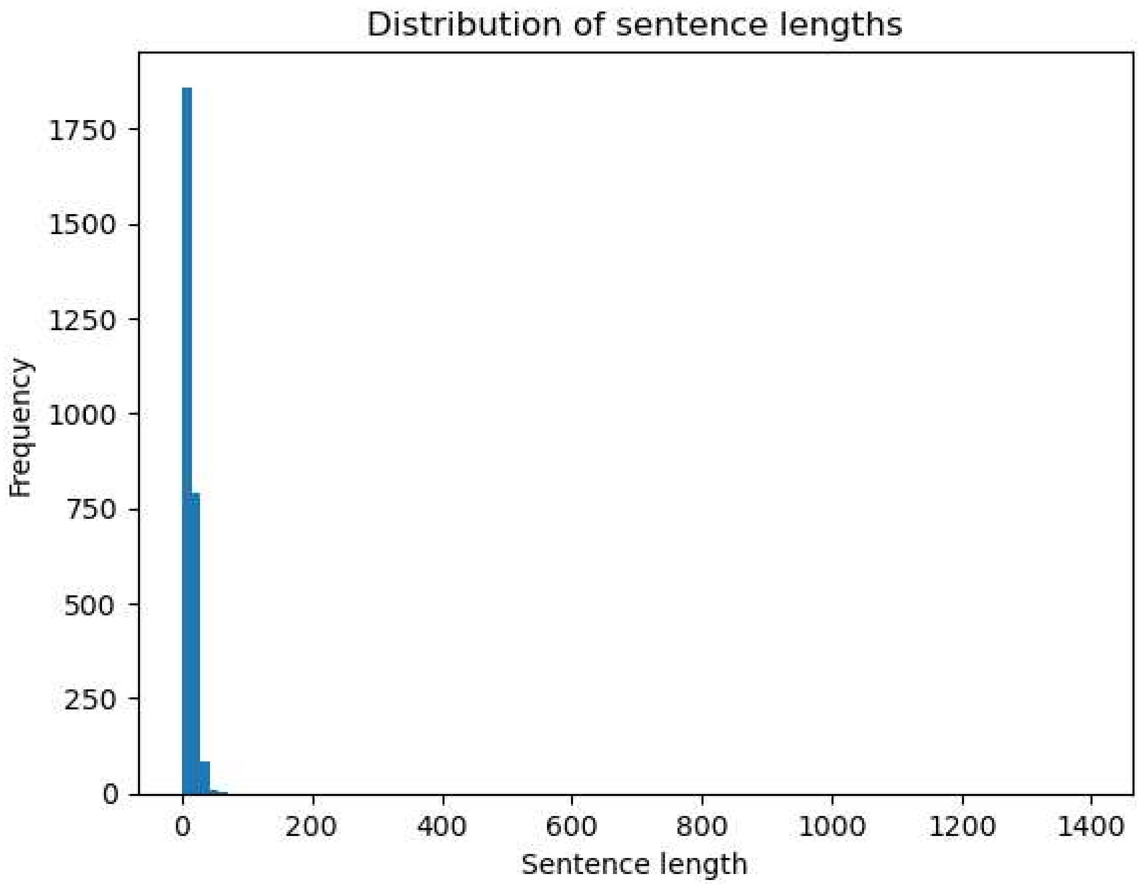
Looking up length of sentences in our data to determine sequence length setting in model

In [6]:
```
commentary_length = []
for char_len in charlookup:
    commentary_length.append(len(char_len.split(' ')))
commentary_max = np.max(commentary_length)
commentary_min = np.min(commentary_length)
commentary_median = np.median(commentary_length)
print("The max length of our sequences would be: ", commentary_max)
print("The min length of our sequences would be: ", commentary_min)
print("The median length of our sequences would be: ", commentary_median)
```

```
The max length of our sequences would be:  1393
The min length of our sequences would be:  1
The median length of our sequences would be:  11.0
```

In [7]:

```
plt.hist(commentary_length, bins=100)
plt.title("Distribution of sentence lengths")
plt.xlabel("Sentence length")
plt.ylabel("Frequency")
plt.show()
```



Building our RNN (Recurrent Neural Network) model using LSTM (Long short-term memory) & Splitting the data

In [8]:

```
stop_words = stopwords.words('english')
lemmatizer = WordNetLemmatizer()

tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")
tokenizer.fit_on_texts(df_a.review) # tokenize the entire review column
sequences = tokenizer.texts_to_sequences(df_a.review) # generate sequences for the ent

# Pad the sequences to a maximum length of 50
maxlen = 50
padded_sequences = pad_sequences(sequences, maxlen=maxlen)

# Split the data into train and test sets
train_padded, test_padded, train_labels, test_labels = train_test_split(padded_sequenc

embedding_vector_length = 40
model = Sequential()
model.add(Embedding(5000, embedding_vector_length, input_length=maxlen)) #Word embeddi
```

```python
model.add(Bidirectional(LSTM(50, dropout=.7, recurrent_dropout=0.5))) #dropout helps p
model.add(Dense(1, activation='sigmoid')) #sigmoid best for binary
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy']) #loss
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 40)            200000

 bidirectional (Bidirectiona  (None, 100)              36400
 l)

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 236,501
Trainable params: 236,501
Non-trainable params: 0
_________________________________________________________________
None
```

In [9]:
```python
# Print the size of the vocabulary
vocab_size = len(tokenizer.word_index) + 1
print("Vocabulary size:", vocab_size)
```

```
Vocabulary size: 5273
```

In [10]:
```python
print(padded_sequences[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   28   51    6   59  119   13   73    8  372    7   12   67   12
    2  189  580    4   78   63    5 2268]
```

In [11]:
```python
pd.DataFrame(train_labels).to_csv('train_labels.csv', sep=',', index=False)
pd.DataFrame(test_labels).to_csv('test_labels.csv', sep=',', index=False)
pd.DataFrame(train_padded).to_csv('train_padded.csv', sep=',', index=False)
pd.DataFrame(test_padded).to_csv('test_padded.csv', sep=',', index=False)
pd.DataFrame(df_a).to_csv('df_a_cleaned.csv', sep=',', index=False)
```

In [12]:
```python
early_stopping = EarlyStopping(
    monitor='val_loss', patience=3, verbose=1, restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss', factor=0.2, patience=1, verbose=1, min_lr=1e-6)

history = model.fit(train_padded, train_labels,
                    validation_data=(test_padded, test_labels),
                    epochs=25, batch_size=128,
                    callbacks=[early_stopping, reduce_lr])
```

```
Epoch 1/25
18/18 [==============================] - 7s 143ms/step - loss: 0.6926 - accuracy: 0.5
118 - val_loss: 0.6945 - val_accuracy: 0.4709 - lr: 0.0010
Epoch 2/25
18/18 [==============================] - 2s 118ms/step - loss: 0.6865 - accuracy: 0.5
460 - val_loss: 0.6887 - val_accuracy: 0.4800 - lr: 0.0010
Epoch 3/25
18/18 [==============================] - 2s 131ms/step - loss: 0.6611 - accuracy: 0.6
169 - val_loss: 0.6489 - val_accuracy: 0.6873 - lr: 0.0010
Epoch 4/25
18/18 [==============================] - 2s 120ms/step - loss: 0.6048 - accuracy: 0.7
443 - val_loss: 0.6134 - val_accuracy: 0.7255 - lr: 0.0010
Epoch 5/25
18/18 [==============================] - 2s 126ms/step - loss: 0.5073 - accuracy: 0.7
898 - val_loss: 0.5237 - val_accuracy: 0.7764 - lr: 0.0010
Epoch 6/25
18/18 [==============================] - 2s 127ms/step - loss: 0.4303 - accuracy: 0.8
308 - val_loss: 0.4840 - val_accuracy: 0.8036 - lr: 0.0010
Epoch 7/25
18/18 [==============================] - 2s 126ms/step - loss: 0.3571 - accuracy: 0.8
744 - val_loss: 0.4536 - val_accuracy: 0.8055 - lr: 0.0010
Epoch 8/25
18/18 [==============================] - 2s 123ms/step - loss: 0.2806 - accuracy: 0.9
008 - val_loss: 0.4402 - val_accuracy: 0.8164 - lr: 0.0010
Epoch 9/25
18/18 [==============================] - 2s 124ms/step - loss: 0.2412 - accuracy: 0.9
222 - val_loss: 0.4281 - val_accuracy: 0.8327 - lr: 0.0010
Epoch 10/25
18/18 [==============================] - ETA: 0s - loss: 0.2069 - accuracy: 0.9372
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
18/18 [==============================] - 2s 124ms/step - loss: 0.2069 - accuracy: 0.9
372 - val_loss: 0.4312 - val_accuracy: 0.8436 - lr: 0.0010
Epoch 11/25
18/18 [==============================] - ETA: 0s - loss: 0.1799 - accuracy: 0.9445
Epoch 11: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
18/18 [==============================] - 2s 122ms/step - loss: 0.1799 - accuracy: 0.9
445 - val_loss: 0.4489 - val_accuracy: 0.8400 - lr: 2.0000e-04
Epoch 12/25
18/18 [==============================] - ETA: 0s - loss: 0.1678 - accuracy: 0.9486Res
toring model weights from the end of the best epoch: 9.

Epoch 12: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
18/18 [==============================] - 2s 126ms/step - loss: 0.1678 - accuracy: 0.9
486 - val_loss: 0.4426 - val_accuracy: 0.8455 - lr: 4.0000e-05
Epoch 12: early stopping
```

Evaluating the model

In [13]:
```
model.evaluate(test_padded, test_labels)
```

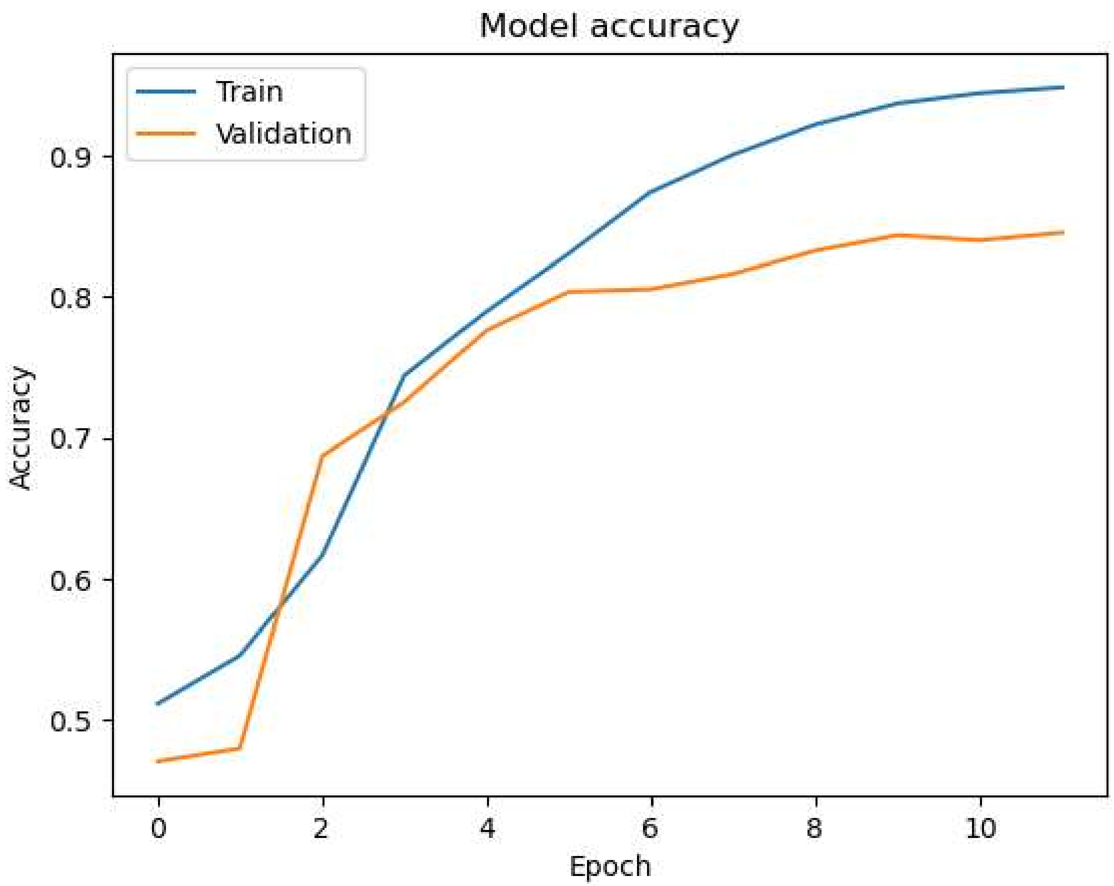
```
18/18 [==============================] - 0s 9ms/step - loss: 0.4281 - accuracy: 0.832
7
```

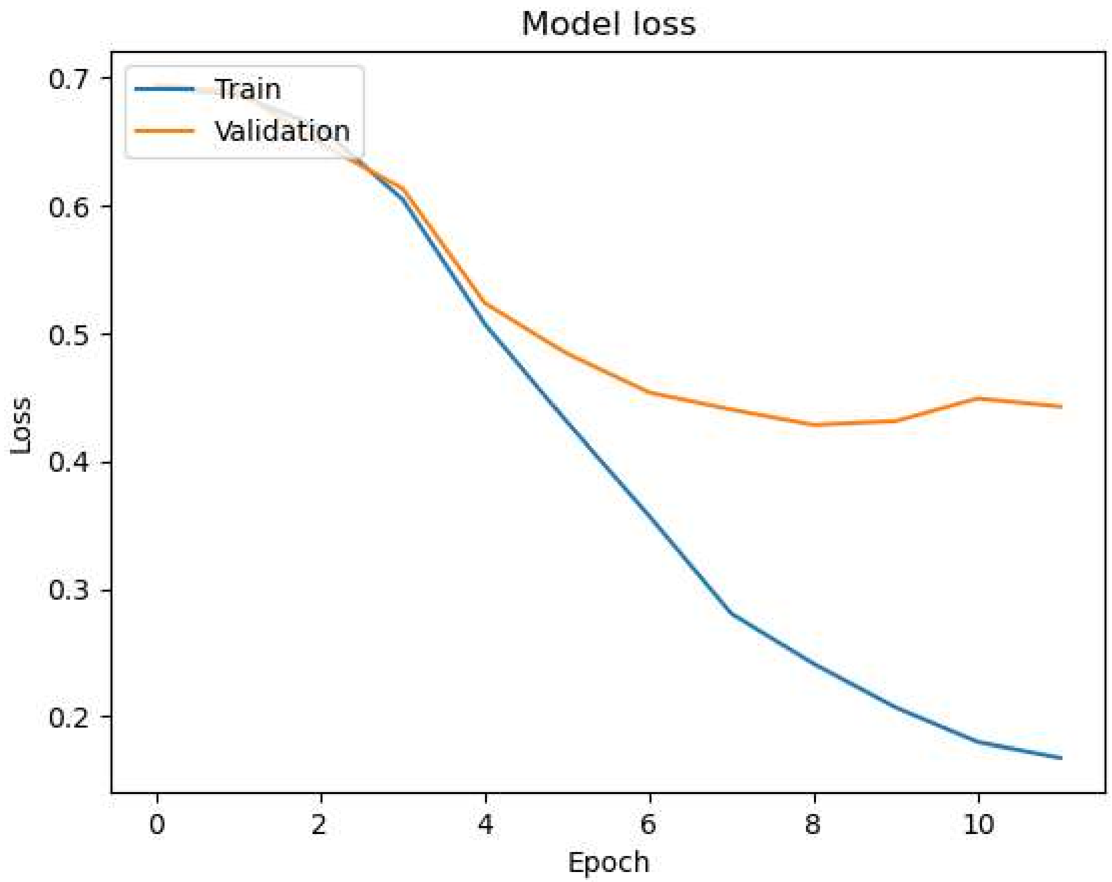
Out[13]:
```
[0.4281434118747711, 0.8327272534370422]
```

In [14]:
```
# Plot the training history
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
```

```
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```





In [15]:
```
loss, accuracy = model.evaluate(test_padded, test_labels)
print(f'Test loss: {loss:.3f}')
print(f'Test accuracy: {accuracy:.3f}')
```

```
18/18 [==============================] - 0s 8ms/step - loss: 0.4281 - accuracy: 0.832
7
Test loss: 0.428
Test accuracy: 0.833
```

Example of model predicting a new review

In [16]:
```
new_review = 'Example review - this product is amazing'
new_review = re.sub("[^a-zA-Z]"," ",new_review)
new_review = new_review.lower()
new_review = nltk.word_tokenize(new_review)
new_review = [lemmatizer.lemmatize(word) for word in new_review]
new_review = [word for word in new_review if not word in stop_words]
new_review = " ".join(new_review)

new_sequence = tokenizer.texts_to_sequences([new_review])
new_padded = pad_sequences(new_sequence, maxlen=maxlen)

prediction = model.predict(new_padded)
if prediction > 0.5:
    print('Positive review')
else:
    print('Negative review')
```

```
1/1 [==============================] - 0s 316ms/step
Positive review
```

In [17]:

```
#saving the model
model.save('DeepNeuralNetwork_RNN_Model.h5')
```

In [ ]:

In [18]:

```
#Using in a pipeline (for future reference)
#pipeline example
#import re
#import nltk
#from nltk.corpus import stopwords
#from nltk.stem import WordNetLemmatizer
#from sklearn.base import BaseEstimator, TransformerMixin
#from sklearn.pipeline import Pipeline

# Custom transformer to preprocess text data
#class TextPreprocessor(BaseEstimator, TransformerMixin):
#    def __init__(self, stop_words=None):
#        self.stop_words = stop_words or stopwords.words('english')
#        self.lemmatizer = WordNetLemmatizer()

#    def fit(self, X, y=None):
#        return self

#    def transform(self, X, y=None):
#        X = X.str.lower()
#        X = X.apply(lambda x: re.sub("[^a-zA-Z]"," ", x))
#       X = X.apply(nltk.word_tokenize)
#        X = X.apply(lambda x: [self.lemmatizer.lemmatize(word) for word in x])
#        X = X.apply(lambda x: [word for word in x if word not in self.stop_words])
#        X = X.apply(lambda x: ' '.join(x))
#        return X
# Example usage
#text_pipeline = Pipeline([
#    ('preprocessor', TextPreprocessor()),
#])
#description_list = text_pipeline.fit_transform(df_a.review)
```