Sentiment Analysis of UCI Data

Mike Mattinson

Western Governors University

Advanced Data Analytics – D213

Task 2: Sentiment Analysis

Dr. Festus Elleh

September 7, 2022

Revision 3

Abstract **2**
Part I. Research Question **3**
Part II. Data Preparation **5**
Part III. Network Architecture **11**
(4) Parameters **12**
Part IV. Model Evaluation **14**
Part V. Summary and Recommendations **16**
Part VI. Reporting **18**
References **19**
Tables **22**
Figures **39**
Appendix A Python Code **52**

Abstract

UCI sentiment will be analyzed and modeled using NLTK/NLP methods to classify positive or negative sentiment from combined Amazon, IMDB, Yelp dataset. Total number of records is 2,700, somewhat small for a robust model, but adequate for a simple model. Keras sequential neural network using embedding, dropout(0.5), flatten and dense(1) layers. Vocabulary (corpus) size approx. 5,000 words.

Keywords: Sentiment Analysis. NLTK. NLP. TensorFlow. Binary Classification. Logistic Classification. Keras Sequential. Embedding.

Part I. Research Question

Section A. Describe the purpose of this data analysis by doing the following:

A1. Summarize one research question that you will answer using neural network models and NLP techniques. Be sure the research question is relevant to a real-world organizational situation and sentiment analysis captured in your chosen dataset.

Can customer unstructured data reviews be used to model positive or negative sentiment?

A2. Define the objectives or goals of the data analysis. Be sure the objectives or goals are reasonable within the scope of the research question and are represented in the available data.

Use Keras neural network to model and predict positive and negative sentiment using a combined Amazon, IMDB and Yelp customer unstructured review dataset.

A3. Identify a type of neural network capable of performing a text classification task that can be trained to produce useful predictions on text sequences on the selected data set.

Keras sequential neural network using the following layers:

- Embedding layer
- Flattening layer
- Dense (1) layer

Using the following Python packages:

- import tensorflow as tf

- from tensorflow import keras
- from sklearn.model_selection import train_test_split as tts
- from numpy import array
- from keras import models
- from keras import layers
- from keras import regularizers
- from sklearn.model_selection import train_test_split
- from keras.preprocessing.text import Tokenizer
- from keras_preprocessing.sequence import pad_sequences
- import wordcloud
- from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
- import nltk
- from nltk.corpus import stopwords
- from nltk.tokenize import word_tokenize
- from nltk.probability import FreqDist
- from nltk.stem import WordNetLemmatizer

Here are the versions of some of the important packages:

```
print('tensorflow ver: {}'.format(tf.__version__))
print('nltk ver: {}'.format(nltk.__version__))
print('wordcloud ver: {}'.format(wordcloud.__version__))
print('numpy ver: {}'.format(np.__version__))
print('pandas ver: {}'.format(pd.__version__))
#print('matplotlib ver: {}'.format(plt.__version__))
```

```
tensorflow ver: 2.9.1
nltk ver: 3.7
wordcloud ver: 1.8.2.2
numpy ver: 1.23.2
pandas ver: 1.4.3
```

Part II. Data Preparation

Section B. Summarize data cleaning process by doing the following:

B1. Perform exploratory data analysis on the chosen dataset, and include an explanation of each of the following elements:

(1) presence of **unusual characters** (e.g., emojis, non-English characters, etc.).

Using the **remove_punctuation** functiton to remove sentence punctuation :

- "?"
- "."
- ";"
- ":"
- "!"
- ""
- ','

(2) **vocabulary size**. Used the length of the lokenizer.word_index to determine that the total vocabulary size is 4,425. This is identified in the model as the variable "**vocab_size**"

(3) **proposed word embedding length**. The optimum word embedding length is identified in the model as the variable "**input_dim**" and is set to be the vocabulary size described above, input_dim = 4,425.

(4) statistical justification for the chosen **maximum sequence length**. The optimum maximum sequence length was determined through observations to be 256 and is identified in the model as the variable “**maxlen**”.

B2. Describe the goals of the tokenization process, including any code generated and packages that are used to normalize text during the tokenization process.

(1) Tokenize sentences “**First Tokenization**”. In order for the stopwords, lemmazation, and infrequent words analysis to be effective, the initial data of sentence/text structure is tokenized into an array of words.

(2) Tokenize words to numbers “**Second Tokenization**”. Then, just before the model is defined, the second tokenization is used to transform the word structure into an array of numbers. The model requires numbers to be able to model data.

B3. Explain the padding process used to standardize the length of sequences, including the following in your explanation:

(1) **Padding**. if the padding occurs before or after the text sequence. The padding is applied after the second tokenization based on the “maxlen” variable.

(2) a **screenshot** of a single padded sequence. Here is a screenshot of a single padded sequence:

```
Out[24]: array([[ 736,  807,  183, 2854,  582,  773,    1,    8,    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],
               [ 780,  114,  108,   34,  313,  233,    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],
```

B4. Identify how many categories of sentiment will be used and an activation function for the final dense layer of the network.

(1) **sentiment categories**. There are two (2) sentiment categories, positive sentiment indicated by an integer value of 1 and negative sentiment indicated by an integer value of 0.

(2) **activation function**. The activation function used in the model is 'sigmoid'. The sigmoid function is a simple case of the softmax multi-classification function. The sigmoid function takes any real number as input and outputs a value in the range of 0 to 1, the larger the input value, the closer the output will be to 1.

B5. Explain the steps used to prepare the data for analysis, including the size of the training, validation, and test set split.

Step 1. **Combine Data**. Used pandas concat() function to combined the three (3) datasets.

Step 2. **Punctuation**. Used python join() function to remove a given list of punctuation characters.

Step 3. **Lowercase**. Used python's lower() function to change the case of all text to lowercase.

Step 4. **Word Tokenizer**. Used nltk's RegexpTokenizer() function to change the sentence structure to array of individual words.

Step 5. **Stopwords**. Used nltk's stopwords.words("english") appended with a short list of my own stopwords, to go through and remove those words from the data.

Step 6. **Remove Infrequent Words**. Used nltk's FreqDist() to compile and count all of words in the data. Then I removed all words based on a cutoff value. In this case, the cutoff value was set at 1, I did not want to remove any words from the already small dataset.

Step 7. **Lemmatize**. Used nltk's WordNetLemmatizer() function to standardize the word tense.

B6. Provide a copy of the prepared dataset.

See Table 10. A copy of prepared dataset is attached to submission and is located in the "tables" folder.

5 export clean data

In [18]:

```
# review what the data looks like after cleaning
print('{}\n{}'.format(df.info(), df.shape))
df.sample(3, random_state=0) # 5 random (0) rows of data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2748 entries, 0 to 2747
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   text               2748 non-null   object
 1   label              2748 non-null   int64
 2   text_token         2748 non-null   object
 3   text_string        2748 non-null   object
 4   text_string_fdist  2748 non-null   object
 5   text_string_lem    2748 non-null   object
dtypes: int64(1), object(5)
memory usage: 128.9+ KB
None
(2748, 6)
```

Out[18]:

	text	label	text_token	text_string	text_string_fdist	text_string_lem
1801	they have horrible attitudes towards customers...	0	[horrible, attitudes, towards, customers, talk...	horrible attitudes towards customers talk one ...	horrible attitudes towards customers talk one ...	horrible attitudes towards customers talk one ...
1590	10/10	1	[10, 10]			
2382	ordered burger rare came in we'll done	0	[ordered, burger, rare, came, done]	ordered burger rare came done	ordered burger rare came done	ordered burger rare came done

In [19]:

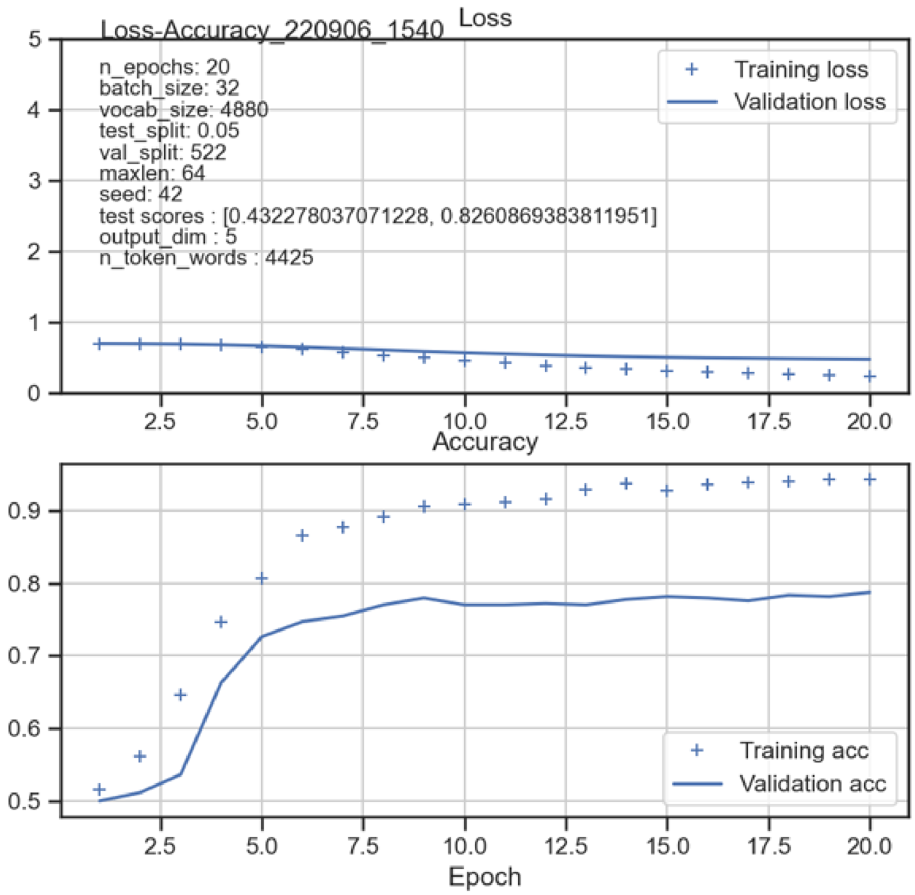
```
# export clean data
f = 'tables\clean.csv'
df.to_csv(f, index=True, header=True)
```

Part III. Network Architecture

Section C. Describe the type of network used by doing the following:

C1. Provide the output of the model summary of the function from TensorFlow.

See Figure 4. The final “best” model summary is as follows:




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 64, 5)             24400

 dropout (Dropout)           (None, 64, 5)             0

 flatten (Flatten)           (None, 320)               0

 dense (Dense)               (None, 1)                 321

=================================================================
Total params: 24,721
Trainable params: 24,721
Non-trainable params: 0
_________________________________________________________________
```

C2. Discuss the number of layers, the type of layers, and total number of parameters.

(1) **Embedding** layer. The embedding layer is used to convert each word to a vector of defined size. “Embedding layer enables us to convert each word into a fixed length vector of defined size. The resultant vector is a dense one with having real values instead of just 0’s and 1’s. The fixed length of word vectors helps us to represent words in a better way along with reduced dimensions.” (Saxena, 2020)

(2) **Flattening** layer. The flattening layer is used to reduce the dimension and shape of the input layer.

(3) **Dense** layer. The dense layer is used to combine all of the available neurons in the model and shape the final output of the model.

(4) **Parameters**. The parameters are sub-divided into trainable and non-trainable parameters. Depending on the vocabulary size, the tokenized word lengths and batch size, each model can have between thousands to millions of parameters. In this analysis, all of the parameters of all the candidate models were “trainable”. See Figure 5 for the number of parameters for each candidate model and Figure 6 for the number of parameters in the final “best” model.

C3. Justify the choice of hyperparameters, including the following elements:

(1) **activation functions**. The sigmod activation function is commonly used for binary classification models. The activation function is specified in the dense layer.

(2) **number of nodes per layer**. The model summary shows the number of parameters associated with each layer.

(3) **loss function**. The loss function for the model is "binary crossentropy". The loss function is specified in the model.compile() code as follows:

```
In [28]:    model.compile(
                        optimizer='adam',
                        loss='binary_crossentropy',
                        metrics=['acc'])
```

(4) **optimizer**. The optimizer is "adam" and is also specified in the compile portion of the code as seen above.

(5) **stopping criteria**. Not used. I ran multiple models of varying number of epochs. Then, looked at the outcome and made updates to the subsequent model based on those observations.

(6) **evaluation metric**. The primary evaluation metric is the Accuracy percentage. The metric is obtained when running the test data through the prediction model.

Part IV. Model Evaluation

Section D. Evaluate the model training process and its relevant outcomes by doing the following:

D1. Discuss the impact of using stopping criteria instead of defining the number of epochs, including a screenshot showing the final training epoch.

All of the candidate and final models were run using a specific number of epochs. All of the models, include some of the more advanced models, all ran in just around 1-5 minutes. Because of the size of the data and simplicity of the model, no stopping criteria as used.

D2. Provide visualizations of the model's training process, including a line graph of the loss and chosen evaluation metric.

There are a number of model visualizations included with Figure 3. There is also an "Excel" table that summarizes some of the critical parameters. The "best model" and final model is included in Figure 4.

D3. Assess the fitness of the model and any measures taken to address overfitting.

Most all of the candidate models show some overfitting. Many different parameters and models were attempted in order to minimize the overfitting. Some models where attempted where the number of layers and neurons were increased, but in the end the simplest model with minimum number of neurons yielded the best results. In the best model, Figure 6, the number of epochs were reduced to approximately 10 epochs and this was sufficient to yield approximately 75% accuracy on test data.

After some time, I determined that the only way to get better metrics and reduce overfitting was to add a drop layer. With the dropout added, the overall accuracy increased, and the overfitting was reduced. The model summary showing the dropout layer:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 64, 5)             24400

 dropout (Dropout)           (None, 64, 5)             0

 flatten (Flatten)           (None, 320)               0

 dense (Dense)               (None, 1)                 321

=================================================================
Total params: 24,721
Trainable params: 24,721
Non-trainable params: 0
_________________________________________________________________
```

D4. Discuss the predictive accuracy of the trained network.

The final model yielded 82.6% accuracy on the testing data.

Part V. Summary and Recommendations

Section E. Provide the code used to save the trained network within the neural network.

“There are two formats you can use to save an entire model to disk: the TensorFlow SavedModel format, and the older Keras H5 format. The recommended format is SavedModel. It is the default when you use model.save(). (TensorFlow.org, 2022)“ I used the default format to save the model as follows:

```
In [31]:  # save model in SavedModel format
          # prior to saving the model, you need to compile the model
          from datetime import datetime
          now = datetime.now() # current date and time
          date_time_stamp = now.strftime("_%y%m%d_%H%M")
          model.save('models/final' + date_time_stamp)

          INFO:tensorflow:Assets written to: models/final_220906_1520\asset
          s
```

Section F. Discuss the functionality of your neural network, including the impact of the network architecture.

A lot of models were executed with and without dropout and in the end, the performance of the neural network (accuracy 82.6%) was really not that much better than the very simple logistic regression model (accuracy 81%).

The neural network models performed better with dropout layer added (accuracy 82.6%) than without dropout (accuracy 80%)

The combined dataset was relatively small, so the impact of this model design was negligible.

Section G. Recommend a course of action based on your results.

Model training and performance accuracy should be able to achieve better results using a larger training dataset.

Part VI. Reporting

Section H. Create your neural network using an industry-relevant interactive development environment (e.g., a Jupyter Notebook). Include a PDF or HTML document of your executed notebook presentation.

Attached Jupyter notebook. Also, copy of Python code from the notebook is included in Appendix A.

Section I. List the web sources used to acquire data or segments of third-party code to support the application.

See references.

Section J. Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

See references.

Section K. Demonstrate professional communication in the content and presentation of your submission.

References

Baka, B. (2017). *Python Data Structures and Algorithms: Improve application performance with graphs, stacks, and queues.* Packt Publishing - ebooks Account.

Bruce, P. C., Gedeck, P., Shmueli, G., & Patel, N. R. (2019). *Data Mining for Business Analytics Concepts, Techniques and Applications in Python.* Wiley & Sons, Incorporated, John.

Bruce, P., Bruce, A., & Gedeck, P. (2020). *Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python.* O'Reilly Media Inc.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms.* The MIT Press.

Daityari, S. (2019). How To Perform Sentiment Analysis in Python 3 Using the Natural Language Toolkit (NLTK) | DigitalOcean. *How To Perform Sentiment Analysis in Python 3 Using the Natural Language Toolkit (NLTK) | DigitalOcean*. Retrieved July 24, 2022, from https://www.digitalocean.com/community/tutorials/how-to-perform-sentiment-analysis-in-python-3-using-the-natural-language-toolkit-nltk

Fenner, M. (2018). *Machine Learning with Python for Everyone.* Addison Wesley.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). *Data Structures and Algorithms in Python.* Wiley.

Griffiths, D. (2009). *Head First Statistics.* O'Reilly Media Inc.

Griffiths, D. (2009). *Head First Statistics.* O'Reilly Media Inc.

Kumari, K. (2021, August). Text Preprocessing techniques for Performing Sentiment Analysis! *Text Preprocessing techniques for Performing Sentiment Analysis!* Retrieved August 12, 2022, from https://www.analyticsvidhya.com/blog/2021/08/text-preprocessing-techniques-for-performing-sentiment-analysis/

Larose, C. D., & Larose, D. T. (2019). *Data Science Using Python and R.* Wiley.

Li, S. (2018, June). A Beginner's Guide on Sentiment Analysis with RNN. *A Beginner's Guide on Sentiment Analysis with RNN*. Retrieved July 27, 2022, from https://towardsdatascience.com/a-beginners-guide-on-sentiment-analysis-with-rnn-9e100627c02e

Massaron, L., & Boschetti, A. (n.d.). *Regression Analysis with Python.* Packt Publishing - ebooks Account.

Mogyorosi, M. (n.d.). Sentiment Analysis: First Steps With Python's NLTK Library – Real Python. *Sentiment Analysis: First Steps With Python's NLTK Library – Real Python*. Retrieved July 26, 2022, from https://realpython.com/python-nltk-sentiment-analysis/

Selvaraj, N. (2020, September). A Beginner's Guide to Sentiment Analysis with Python. *A Beginner's Guide to Sentiment Analysis with Python*. Retrieved July 26, 2022, from https://towardsdatascience.com/a-beginners-guide-to-sentiment-analysis-in-python-95e354ea84f6

Sriniketh, J. (2021, June). Sentiment Analysis using NLTK - A Practical Approach. *Sentiment Analysis using NLTK - A Practical Approach*. Retrieved July 27, 2022, from https://www.analyticsvidhya.com/blog/2021/06/sentiment-analysis-using-nltk-a-practical-approach/

VanderPlas, J. (2016). *In Depth: Principal Component Analysis.* O'Reilly. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

Virahonda, S. (2020, October). An easy tutorial about Sentiment Analysis with Deep Learning and Keras. *An easy tutorial about Sentiment Analysis with Deep Learning and Keras*. Retrieved July 27, 2022, from https://towardsdatascience.com/an-easy-tutorial-about-sentiment-analysis-with-deep-learning-and-keras-2bf52b9cba91

Tables

Table 1 *Raw data* 23

Table 2 *Descriptive Statistics* 24

Table 3 *Dataset after removing Punctuation* 25

Table 4 *Dataset after converting to Lowercase* 26

Table 5 *Dataset after first Tokenization* 27

Table 6 *Dataset after removing Stopwords* 28

Table 7 *Dataset after removing infrequent words* 29

Table 8 *Dataset after applying Lemmatizer* 30

Table 9 *Finding Most Common* 31

Table 10 *Final Cleaned and Prepared Dataset* 32

Table 11 *Finding "good" in negative sentiment* 33

Table 12 *Finding "great" in negative sentiment* 34

Table 13 *Finding "bad" in positive sentiment* 35

Table 14 *Train Test Split* 36

Table 15 *Second Tokenizer – Words (Series) -> Numbers (list of list)* 37

Table 16 *Confusion Matrix from LogisticsRegression Model (Amazon dataset)* 38

Table 1

Raw data

In [2]:

```
# read csv data
amazon = 'data/amazon_cells_labelled.txt'
imdb =  'data/imdb_labelled.txt'
yelp =  'data/yelp_labelled.txt'
colnames=['text', 'label']
amazon_df = pd.read_csv(amazon, sep='\t', names=colnames, header=None)
imdb_df = pd.read_csv(imdb, sep='\t', names=colnames, header=None)
yelp_df = pd.read_csv(yelp, sep='\t', names=colnames, header=None)
df = pd.concat([amazon_df, imdb_df, yelp_df])
df = df.reset_index(drop=True)

print('{}\n{}'.format(df.info(), df.shape))
df.sample(5, random_state=0) # 5 random (0) rows of data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2748 entries, 0 to 2747
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    2748 non-null   object
 1   label   2748 non-null   int64
dtypes: int64(1), object(1)
memory usage: 43.1+ KB
None
(2748, 2)
```

Out[2]:

	text	label
1801	They have horrible attitudes towards customers...	0
1590	10/10	1
2382	Ordered burger rare came in we'll done.	0
2447	Anyways, The food was definitely not filling a...	0
1147	This is actually a very smart movie.	1

Notes. Raw data showing 1,000 data values. The sentiment column ‘score’ is integer value of 1 for positive sentiment and 0 otherwise.

Table 2

Descriptive Statistics

In [39]:

```
# descriptive stattics
print(type(df['label']))
print(df['label'].info())
df.describe()
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
RangeIndex: 2748 entries, 0 to 2747
Series name: label
Non-Null Count  Dtype
--------------  -----
2748 non-null   int64
dtypes: int64(1)
memory usage: 21.6 KB
None
```

Out[39]:

	label
count	2748.000000
mean	0.504367
std	0.500072
min	0.000000
25%	0.000000
50%	1.000000
75%	1.000000
max	1.000000

Notes. The target variable is ‘label’. There are a total of 2,748 records. No missing data.

Table 3

Dataset after removing Punctuation

In [10]:

```
# remove punctuation
def remove_punctuation(text: str) -> str:
    '''remove punctuation from text'''
    final = "".join(u for u in text if u not in (
        "?", ".", ";", ":", "!", '"', ','))
    return final # updated string
print('before: {}'.format(df['text'].loc[0]))
df['text'] = df['text'].apply(remove_punctuation)
print('\nafter: {}'.format(df['text'].loc[0]))
```

```
before: So there is no way for me to plug it in here
in the US unless I go by a converter.

after: So there is no way for me to plug it in here i
n the US unless I go by a converter
```

Notes. Notice the sample showing before and after removing punctuation.

Table 4

Dataset after converting to Lowercase

```
In [11]:
# Lower case
print('before: {}'.format(df['text'].loc[0]))
df['text'] = df['text'].astype(str).str.lower()
print('\nafter: {}'.format(df['text'].loc[0]))

before: So there is no way for me to plug it in here
in the US unless I go by a converter

after: so there is no way for me to plug it in here i
n the us unless i go by a converter
```

Notes. Sample showing before and after converting to lowercase.

Table 5

Dataset after first Tokenization

```
In [23]:  # first tokenization
          from nltk.tokenize import RegexpTokenizer
          regexp = RegexpTokenizer('\w+')
          print('before: {}'.format(df['text'].loc[0]))
          df['text_token']=df['text'].apply(regexp.tokenize)
          print('\nafter: {}'.format(df['text_token'].loc[0]))

          # what is type of the new field
          print('\ntext_token type: {}'.format(type(df['text_tok

          before: so there is no way for me to plug it in here
          in the us unless i go by a converter

          after: ['so', 'there', 'is', 'no', 'way', 'for', 'm
          e', 'to', 'plug', 'it', 'in', 'here', 'in', 'the', 'u
          s', 'unless', 'i', 'go', 'by', 'a', 'converter']

          text_token type: <class 'pandas.core.series.Series'>
```

Notes. df['text_token'] now created as a pandas series of tokenized words that make up the original text.

Table 6

Dataset after removing Stopwords

```
In [20]:  # remove stopwords
          stopwords = nltk.corpus.stopwords.words("english")
          print(stopwords[0:20]) # just first 20 stopwords...
          #my_stopwords = ['https', 'good', 'great', 'bad']
          my_stopwords = ['https']
          stopwords.extend(my_stopwords)
          print('\nbefore: {}'.format(df['text'].loc[0]))
          df['text_token'] = df['text_token'].apply(
              lambda x: [item for item in x if item not in stopw
          print('\nafter: {}'.format(df['text_token'].loc[0]))

          ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'our
          selves', 'you', "you're", "you've", "you'll", "yo
          u'd", 'your', 'yours', 'yourself', 'yourselves', 'h
          e', 'him', 'his']

          before: so there is no way for me to plug it in here
          in the us unless i go by a converter

          after: ['way', 'plug', 'us', 'unless', 'go', 'convert
          er']
```

Notes. Stopwords like [‘i’, ‘the’, ‘in’] etc., are removed and the sample text is displayed showing before and after.

Table 7

Dataset after removing infrequent words

```
In [26]:  # remove infrequent words
df['text_string'] = df['text_token'].apply(
    lambda x: ' '.join([item for item in x if len(item)>2]))
all_words = ' '.join([word for word in df['text_string']])
tokenized_words = nltk.tokenize.word_tokenize(all_words)
from nltk.probability import FreqDist
fdist = FreqDist(tokenized_words)
print(fdist)
cutoff = 1 # drop words occurring less than certain amount
print('\nbefore: {}'.format(df['text'].loc[0]))
df['text_string_fdist'] = df['text_token'].apply(
    lambda x: ' '.join([item for item in x if fdist[item] >= cutoff ]))
print('\nafter (text_string): {}'.format(df['text_string'].loc[0]))
print('\nafter (text_string_fdist): {}'.format(df['text_string_fdist'].loc[0

<FreqDist with 5129 samples and 28066 outcomes>

before: so there is no way for me to plug it in here in the us unless i go
by a converter

after (text_string): there way for plug here the unless converter

after (text_string_fdist): there way for plug here the unless converter
```

Notes. Text updated to remove infrequent words. In this case, the cutoff is set at 1, so there are no words meeting the criteria. The dataset is small and the analysis is limited based on the small size, so I did not want to remove any words from the analysis. But, this code might be helpful in the future when working with much larger datasets.

Table 8

Dataset after applying Lemmatizer

```
In [27]:  # Lemmatize
          wordnet_lem = WordNetLemmatizer()
          print('\nbefore: {}'.format(df['text'].loc[0]))
          df['text_string_lem'] = df['text_string_fdist'].apply(wordnet_lem.lemmatize)
          print('\nafter (text_string_lem): {}'.format(df['text_string_lem'].loc[0]))
```

```
before: so there is no way for me to plug it in here in the us unless i go
by a converter

after (text_string_lem): there way for plug here the unless converter
```

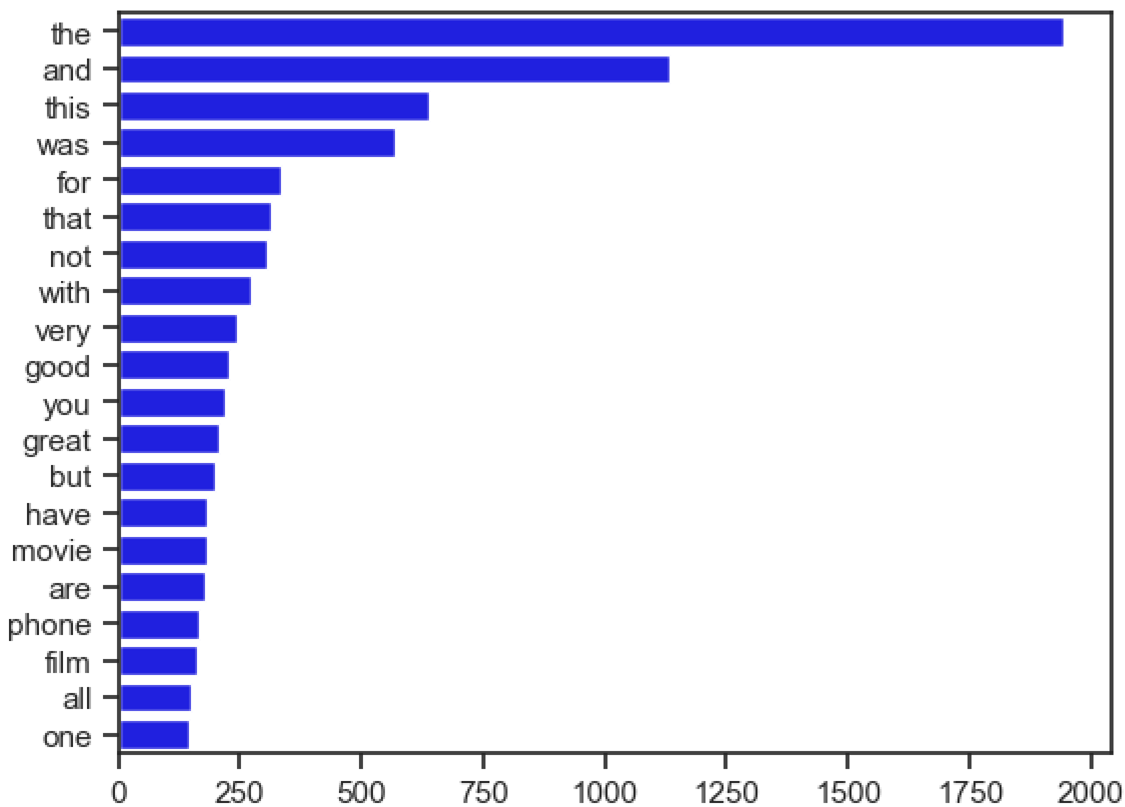
Notes.

Table 9

Finding Most Common

```
In [31]:
# finding most common words
n_common = 20
all_words_lem = ' '.join([word for word in df['text_string_lem']])
words = nltk.word_tokenize(all_words_lem)
fd = FreqDist(words)
top_x_words = fd.most_common(n_common)
fdist = pd.Series(dict(top_x_words)) # data converted to series
import seaborn as sns
sns.set_theme(style="ticks")
sns.barplot(y=fdist.index, x=fdist.values, color='blue');
print(fd.most_common(n_common))
```

```
[('the', 1944), ('and', 1132), ('this', 641), ('was', 571), ('for', 336),
('that', 316), ('not', 306), ('with', 273), ('very', 243), ('good', 229),
('you', 221), ('great', 208), ('but', 200), ('have', 184), ('movie', 181),
('are', 180), ('phone', 165), ('film', 163), ('all', 148), ('one', 145)]
```



Notes. Most common word plotted as histogram.

Table 10

Final Cleaned and Prepared Dataset

In [33]:
```
# review what the data looks like after cleaning
print('{}\n{}'.format(df.info(), df.shape))
df.sample(3, random_state=0) # 5 random (0) rows of data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2748 entries, 0 to 2747
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   text               2748 non-null   object
 1   label              2748 non-null   int64 
 2   text_token         2748 non-null   object
 3   text_string        2748 non-null   object
 4   text_string_fdist  2748 non-null   object
 5   text_string_lem    2748 non-null   object
dtypes: int64(1), object(5)
memory usage: 128.9+ KB
None
(2748, 6)
```

Out[33]:

	text	label	text_token	text_string	text_string_fdist	text_string_lem
1801	they have horrible attitudes towards customers...	0	[they, have, horrible, attitudes, towards, cus...	they have horrible attitudes towards customers...	they have horrible attitudes towards customers...	they have horrible attitudes towards customers...
1590	10/10	1	[10, 10]			
2382	ordered burger rare came in we'll done	0	[ordered, burger, rare, came, in, we, ll, done]	ordered burger rare came done	ordered burger rare came done	ordered burger rare came done

In [38]:
```
# export clean data
f = 'tables\clean.csv'
df.to_csv(f, index=True, header=True)
```

Notes. Final clean data is ready to model and saved in ‘tables’ folder.

Table 11

Finding "good" in negative sentiment

In [4]:

```
# look at 'good' in a negative context
df[(df['text'].str.contains('good') >= 1) & (df['label'] == 0
```

Out[4]:

	text	label
81	Not a good bargain.	0
374	Not a good item.. It worked for a while then s...	0
455	Not good when wearing a hat or sunglasses.	0
563	If you are looking for a good quality Motorola...	0
637	However, BT headsets are currently not good fo...	0
639	Disappointing accessory from a good manufacturer.	0
667	Looks good in the picture, but this case was a...	0
741	Couldn't use the unit with sunglasses, not goo...	0
767	I really wanted the Plantronics 510 to be the ...	0
806	At first I thought I was grtting a good deal a...	0
826	Not as good as I had hoped.	0
848	Not good enough for the price.	0
905	Not nearly as good looking as the AMAZON pictu...	0
1019	The structure of this film is easily the most...	0
1050	The directing and the cinematography aren't qu...	0
1103	It was a good thing that the tickets only cost...	0
1105	This is a bad film, with bad writing, and good...	0
1109	I was left shattered from the experience of wa...	0
1111	I certainly do not mean this distinction in a ...	0
1136	In fact, it's hard to remember that the part ...	0
1149	I love it. \t1\nThe ending is so, SO perfect...	0
1255	Not even good for camp value!	0
1356	It's a shame to see good actors like Thomerson...	0

Notes. The word "Good" was found in 13 rows where the sentiment was negative. It should be removed from the analysis because it will skew the results.

Table 12

Finding "great" in negative sentiment

```
In [5]:  # look at 'great' in a negative context
         df[(df['text'].str.contains('great') >= 1) & (df['label'] ==
```

Out[5]:

	text	label
84	This item worked great, but it broke after 6 m...	0
90	For a product that costs as much as this one d...	0
228	I have had this phone for over a year now, and...	0
355	The loudspeaker option is great, the bumpers w...	0
799	I tried talking real loud but shouting on the ...	0
914	My phone sounded OK (not great - OK), but my ...	0
1019	The structure of this film is easily the most...	0
1072	All in all, a great disappointment.	0
1136	In fact, it's hard to remember that the part ...	0
1283	And, FINALLY, after all that, we get to an end...	0
1326	Full of unconvincing cardboard characters it i...	0
1385	It failed to convey the broad sweep of landsca...	0
2386	After waiting an hour and being seated, I was ...	0

Notes.

Table 13

Finding "bad" in positive sentiment

In [6]:
```
# look at 'bad' in a positive context
df[(df['text'].str.contains('bad') >= 1) & (df['label'] == 1
```

Out[6]:

	text	label
1135	You'll love it! \t1\nThis movie is BAD. \t0\...	1
1245	The last 15 minutes of movie are also not bad ...	1
1379	The film's sole bright spot was Jonah Hill (wh...	1
1596	Predictable, but not a bad watch.	1
1625	I struggle to find anything bad to say about i...	1
1646	With great sound effects, and impressive spec...	1

Notes.

Table 14

Train Test Split

```
In [20]:  # train test split
          X = df['text_string_lem']
          y = df['label']
          seed = 42 # try different seeds
          test_split = 0.25 # 0.2 best so far
          X_train, X_test, y_train, y_test = tts(X, y,
                  test_size=test_split, random_state=seed)
          print(X_train[0:3]) # df['text_string_lem']
          print('X_train shape-type: {}-{}'.format(X_train.shape, type(X_train)))
          print('X_test shape: {}'.format(X_test.shape))
          print('y_train shape-type: {}-{}'.format(y_train.shape, type(y_train)))
          print('y_test shape: {}'.format(y_test.shape))

          350                                jerks phone
          2519    great time family dinner sunday night
          1044                             disappointing
          Name: text_string_lem, dtype: object
          X_train shape-type: (2061,)-<class 'pandas.core.series.Series'>
          X_test shape: (687,)
          y_train shape-type: (2061,)-<class 'pandas.core.series.Series'>
          y_test shape: (687,)
```

Notes. The model ready data is split into training and testing datasets. Notice the model will be using lemmatized text from the data cleaning and processing steps. Also note, all four (4) of the datasets are 'Series'.

Table 15

Second Tokenizer – Words (Series) -> Numbers (list of list)

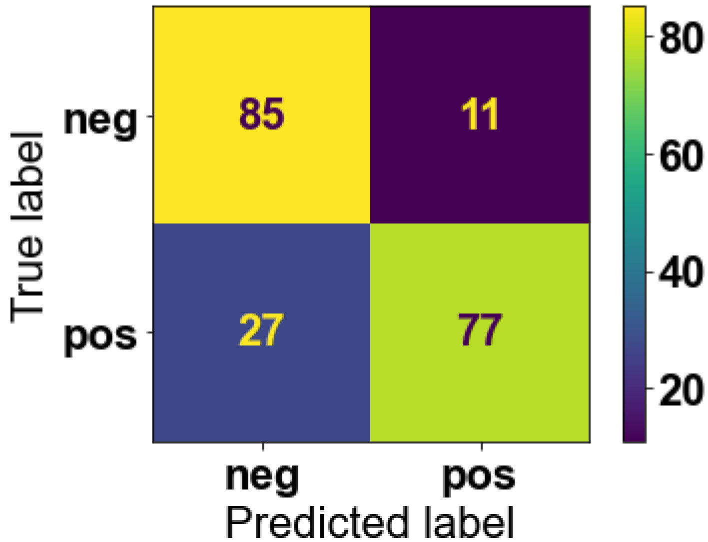
In [21]:

```
# second tokenizer words -> numbers
n_token_words = 5000
tokenizer = Tokenizer(num_words=n_token_words)
print('\nbefore: {}'.format(X_test[:5]))
tokenizer.fit_on_texts(X_train)
print('type(X_train): {}'.format(type(X_train)))
X_train = tokenizer.texts_to_sequences(X_train) # ndarry/df -> list
X_test = tokenizer.texts_to_sequences(X_test)
print('\nafter: {}'.format(X_test[0:10])) # now a list
```

```
before: 2516    close house low key non fancy affordable price...
2642                  stay vegas must get breakfast least
1359    let start problems acting especially lead prof...
1702    bad everyone else involved share crowe level d...
2660    felt insulted disrespected could talk judge an...
Name: text_string_lem, dtype: object
type(X_train): <class 'pandas.core.series.Series'>

after: [[644, 655, 175, 3436, 447, 364, 1, 8], [508, 113, 130, 32, 325, 29
4], [381, 2824, 245, 42, 199, 1320, 12], [12, 236, 405, 604, 1449, 708, 248
9, 21, 98, 29, 4, 350, 688, 695, 285], [291, 1222, 2435, 24, 142, 1494, 13
1, 552, 7], [287, 41, 2106, 153, 249], [544, 1338, 472], [49, 3426, 260, 2
9, 6], [167, 114, 41, 3602], [179, 30, 39, 3701, 2538]]
```

Table 16

Confusion Matrix from LogisticsRegression Model (Amazon dataset)



Results Explained

The Confusion Matrix created has four different quadrants:

True Negative (Top-Left Quadrant)
False Negative (Top-Right Quadrant)
False Positive (Bottom-Left Quadrant)
True Positive (Bottom-Right Quadrant)

```
[[85 11]
 [27 77]]
```

	precision	recall	f1-score	support
0	0.76	0.89	0.82	96
1	0.88	0.74	0.80	104
accuracy			0.81	200
macro avg	0.82	0.81	0.81	200
weighted avg	0.82	0.81	0.81	200

Notes. Overall accuracy is 81%. Early in the assignment, I performed a logistic regression on the Amazon dataset and this was the result.

Figures

Figure 1 *Histogram of Scores* 40

Figure 2 *Wordcloud* 41

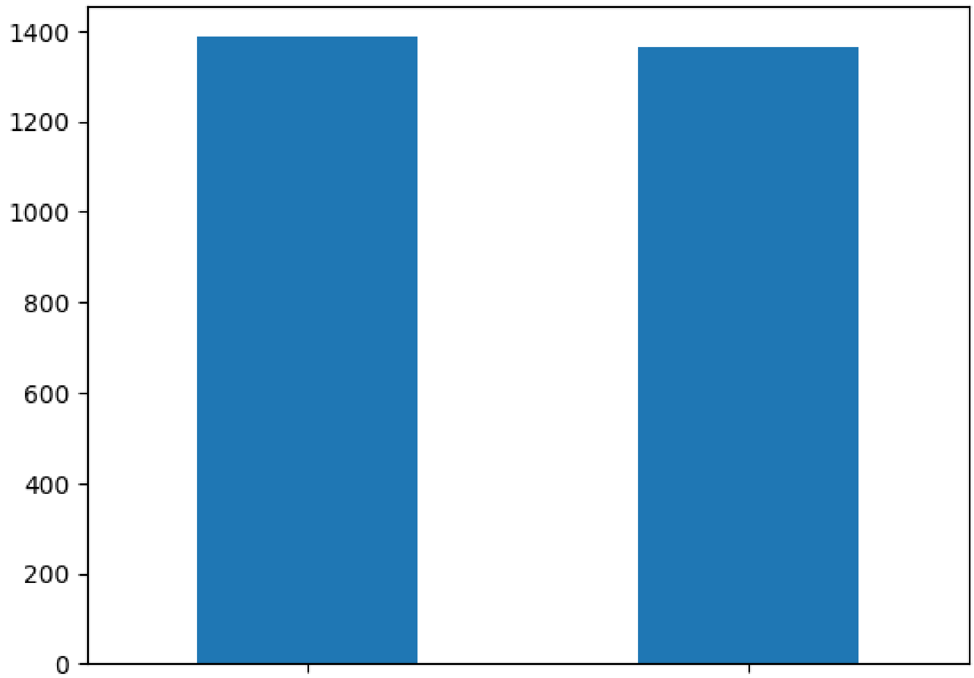
Figure 3 *Candidate Models* 42

Figure 4 *Best Model* 51

Figure 1

Histogram of Scores

```
In [3]:  # plot scores as bar plot
         print(df['label'].value_counts()) # output to notebook
         pd.value_counts(df['label']).plot.bar() # create plot

         1    1386
         0    1362
         Name: label, dtype: int64

Out[3]:  <AxesSubplot:>
```



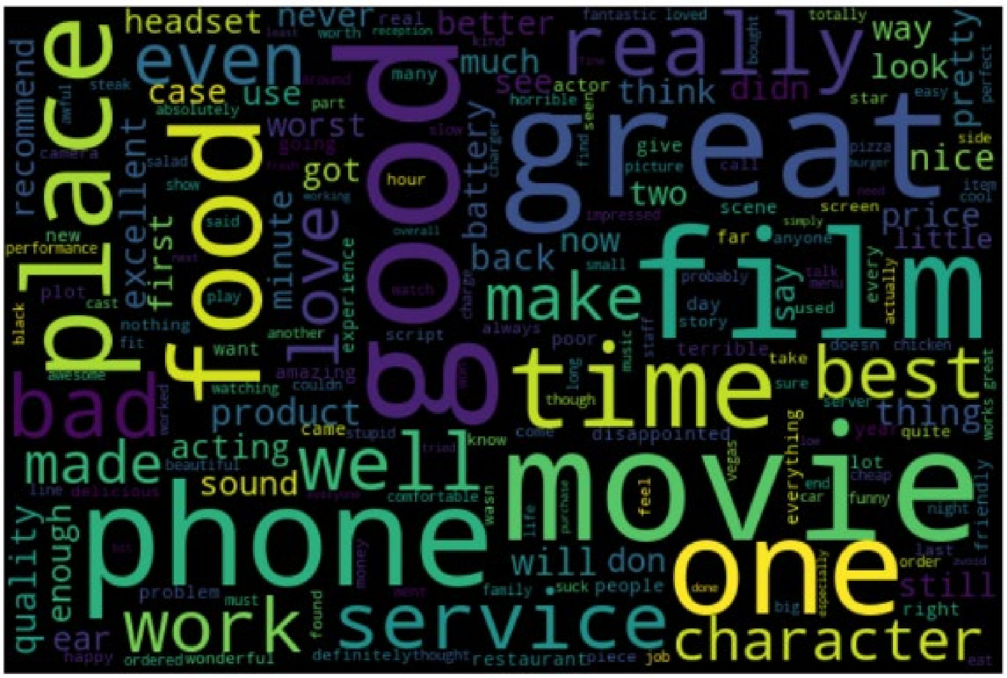
Notes. Histogram plot of raw data. Equally balanced between positive (label=1) and negative (label=0) sentiment.

Figure 2

Wordcloud

In [34]:

```
# wordcloud
wordcloud = WordCloud(width=600,
                      height=400,
                      random_state=2,
                      max_font_size=100).generate(all_words_lem)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off');
```



Notes. A wordcloud of all words regardless. Notice the words “good” and “great” appear in this wordcloud.

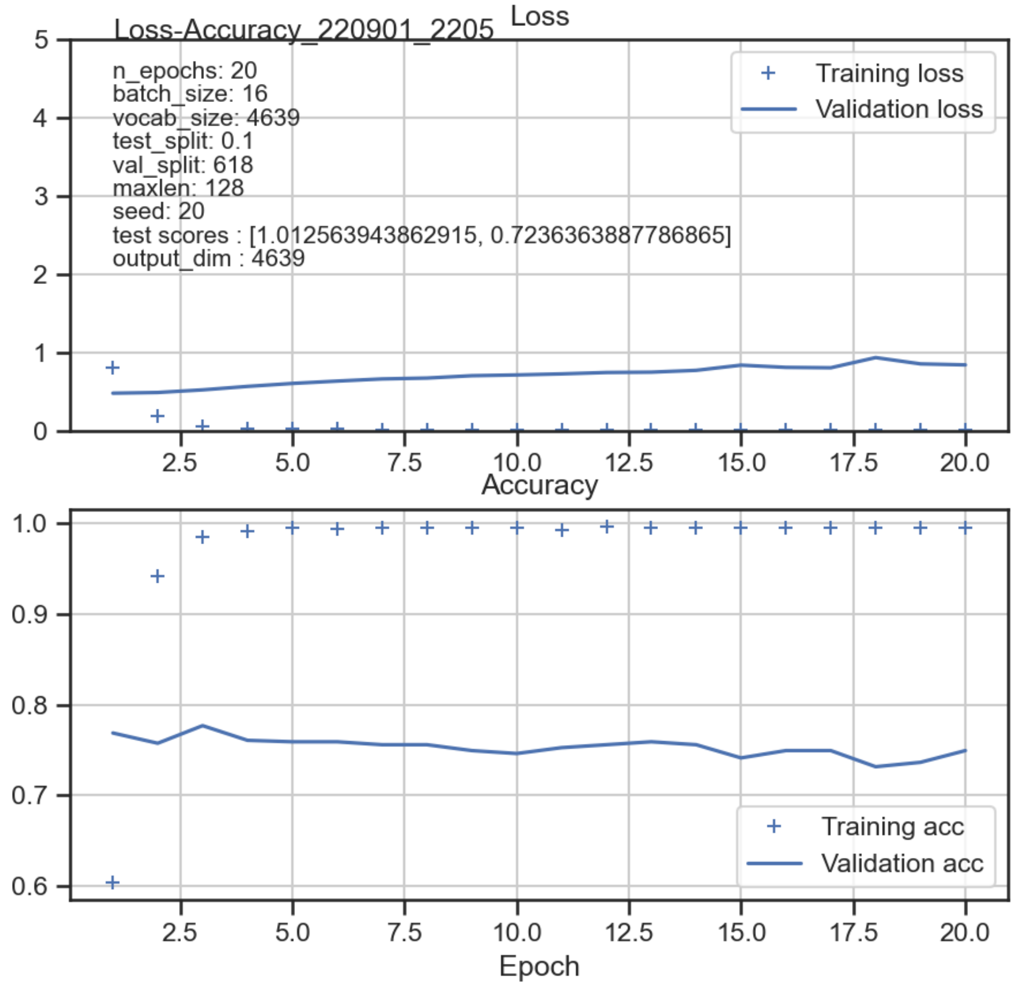
Figure 3

Candidate Models

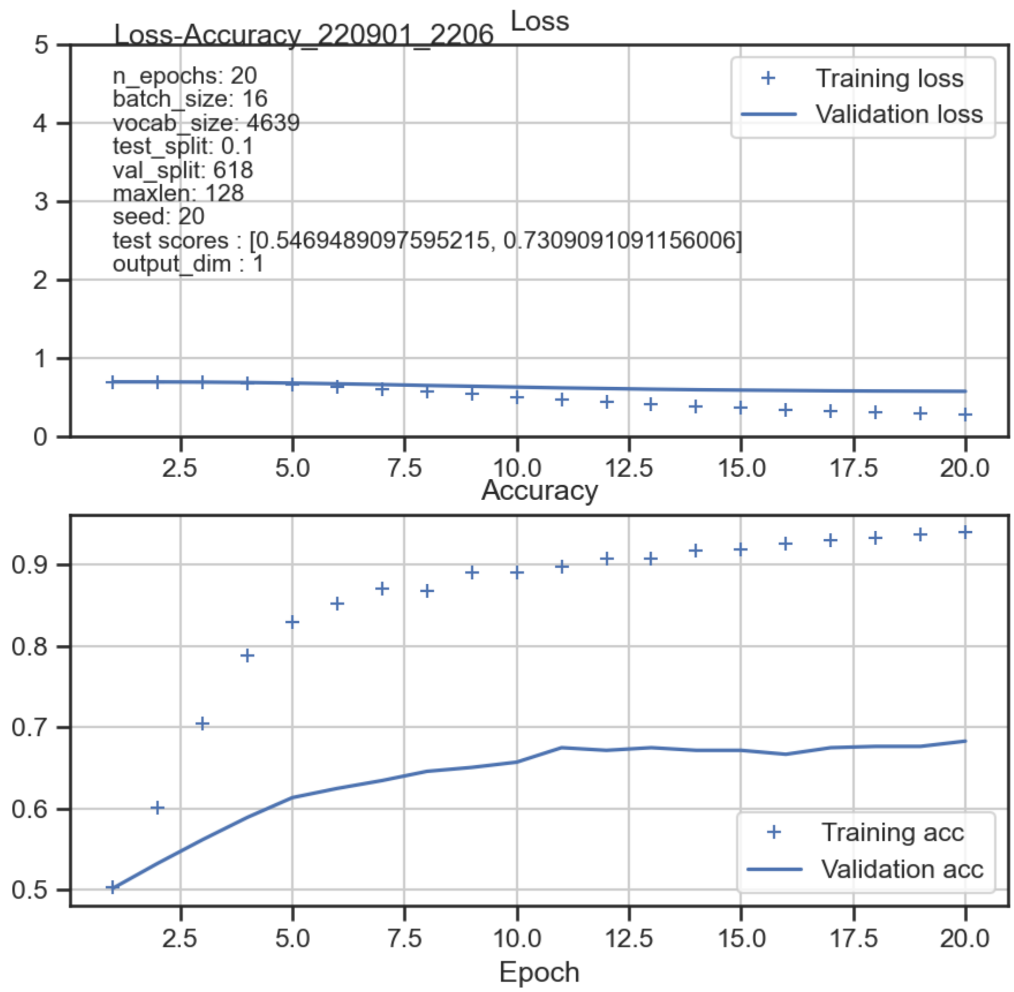
Run ID	Acc %	# epochs	batch size	# token words	seed	test split	drop out	vocab size	output dim	max len
220901_2205	72%	20	16	5,000	20	0.1	No	4,639	4,639	128
220901_2206	73%	20	16	5,000	20	0.1	No	4,639	1	128
220901_2208	80%	30	16	5,000	20	0.1	No	4,639	3	128
220901_2209	76%	30	16	5,000	20	0.2	No	4,363	3	128
220901_2328	76%	30	16	5,000	30	0.3	No	3,907	5	64
220901_2331	74%	30	32	5,000	42	0.25	No	4,231	1,000	128
220906_0931	75%	30	32	5,000	42	0.2	No	4,425	1,000	128
220906_0933	68%	30	32	100	42	0.2	No	4,425	1,000	128
220906_0935	73%	30	32	1,000	42	0.2	No	4,425	1,000	128
220906_0939	75%	30	32	4,000	42	0.2	No	4,425	1,000	128
220906_0940	75%	30	32	10,000	42	0.2	No	4,425	1,000	128
220906_0948	74%	40	64	5,000	42	0.2	No	4,425	5,000	256
220906_1017	74%	40	128	5,000	42	0.2	No	4,425	5,000	256
220906_1131	72%	40	128	4,425	42	0.2	No	3,732	5,000	128
220906_1152	71%	12	64	4,425	42	0.2	No	3,732	4,000	128
220906_1158	75%	20	32	4,425	42	0.2	No	4,425	3,500	64
220906_1203	75%	20	32	4,425	42	0.2	No	4,425	2,500	64

min	68%
max	80%
ave	74%

Notes. Table summarizes some of the best model runs. Cell is highlighted if value is different from the row above it.

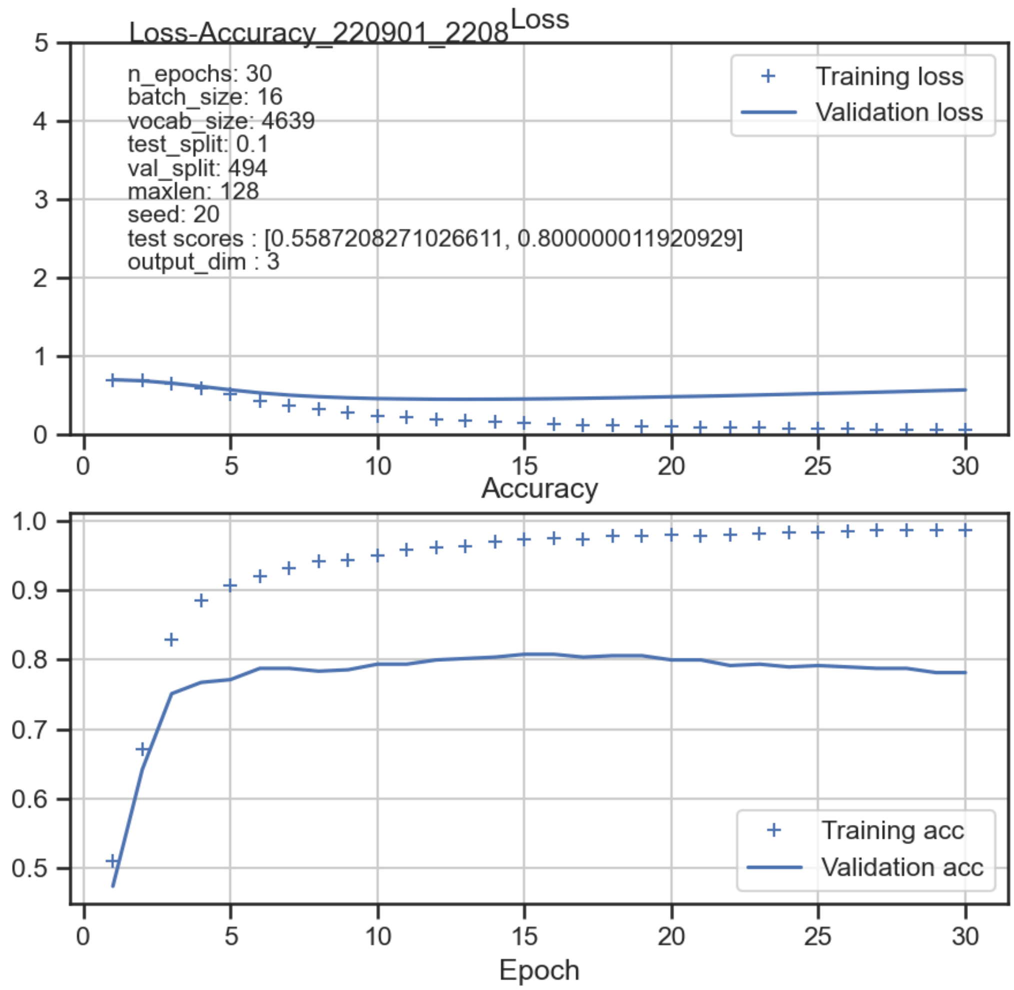



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 128, 4639)         21520321

 flatten (Flatten)           (None, 593792)            0

 dense (Dense)               (None, 1)                 593793

=================================================================
Total params: 22,114,114
Trainable params: 22,114,114
Non-trainable params: 0
_________________________________________________________________
```

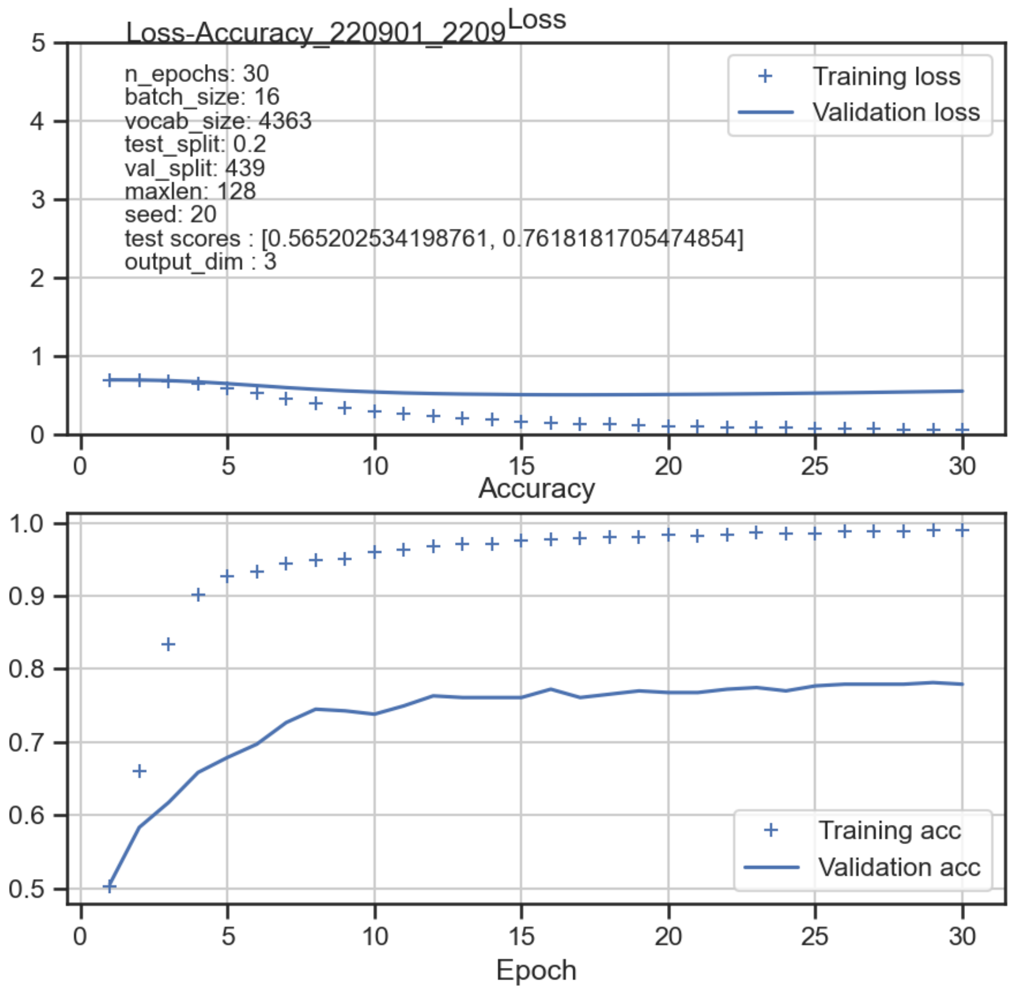



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 128, 1)            4639

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 4,768
Trainable params: 4,768
Non-trainable params: 0
_________________________________________________________________
```

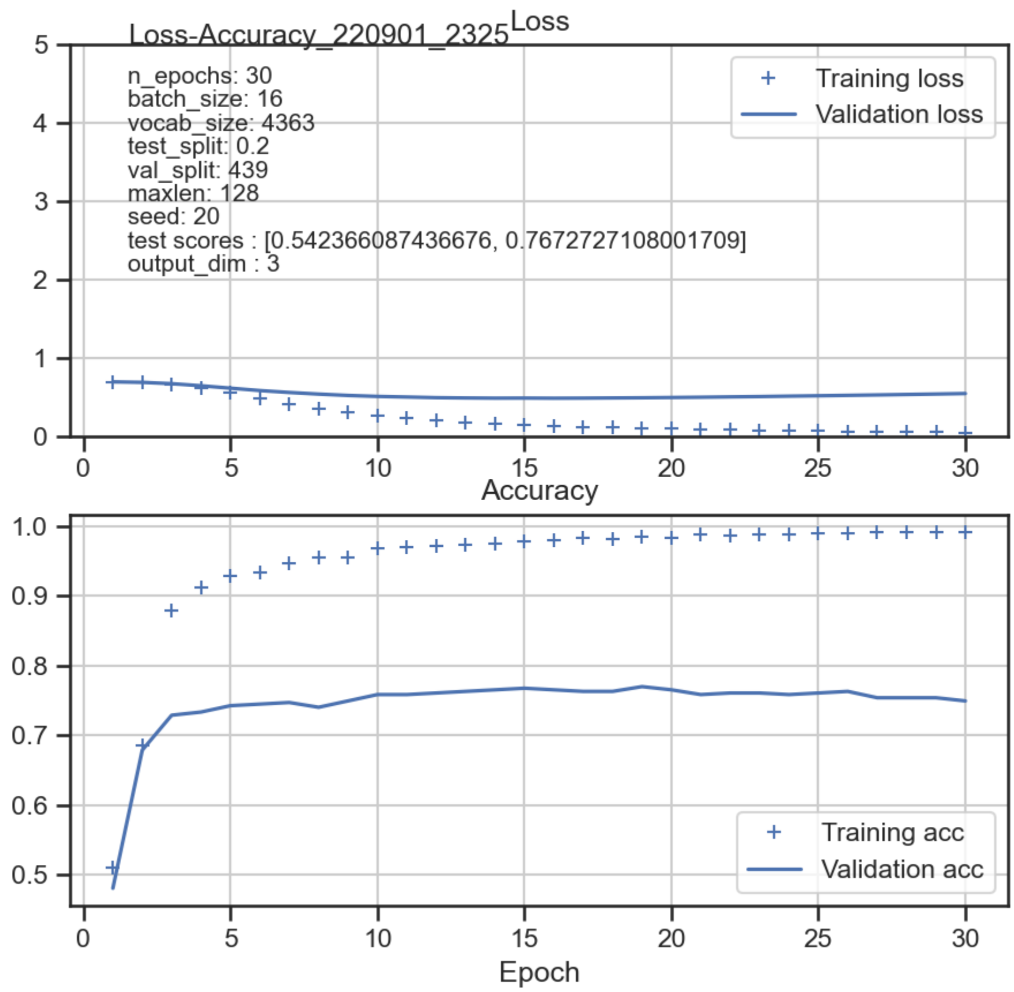
Best Accuracy w/o Dropout



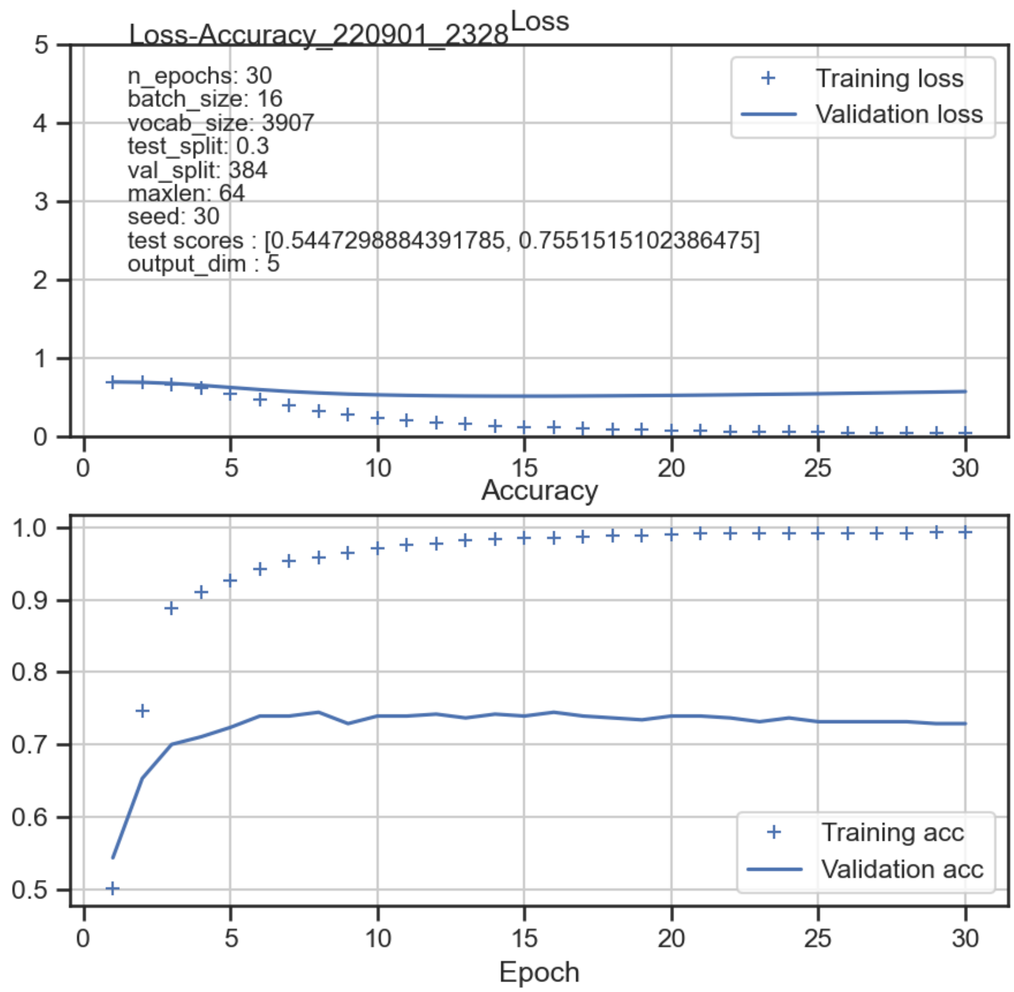

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 128, 3)            13917

 flatten (Flatten)           (None, 384)               0

 dense (Dense)               (None, 1)                 385

=================================================================
Total params: 14,302
Trainable params: 14,302
Non-trainable params: 0
_________________________________________________________________
```




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 128, 3)            13089

 flatten (Flatten)           (None, 384)               0

 dense (Dense)               (None, 1)                 385

=================================================================
Total params: 13,474
Trainable params: 13,474
Non-trainable params: 0
_________________________________________________________________
```



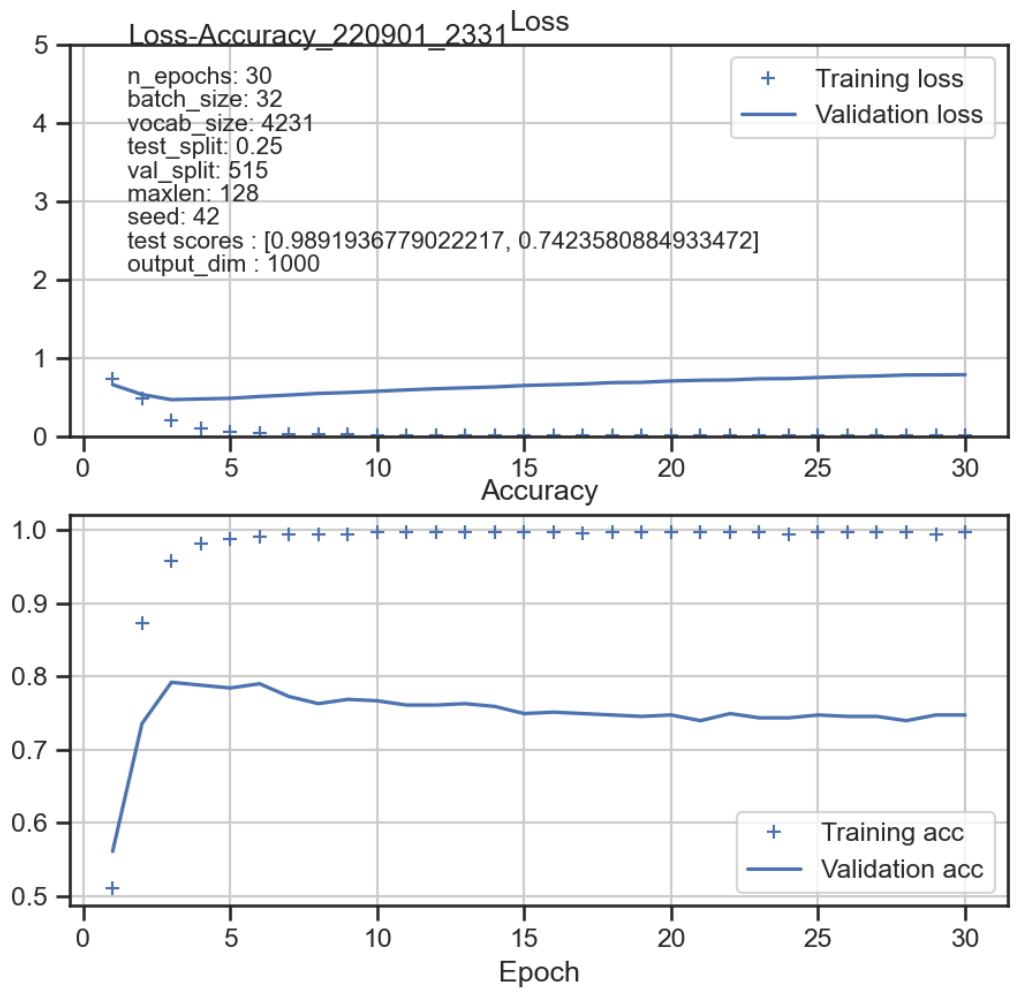

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 128, 3)            13089

 flatten (Flatten)           (None, 384)               0

 dense (Dense)               (None, 1)                 385

=================================================================
Total params: 13,474
Trainable params: 13,474
Non-trainable params: 0
_________________________________________________________________
```



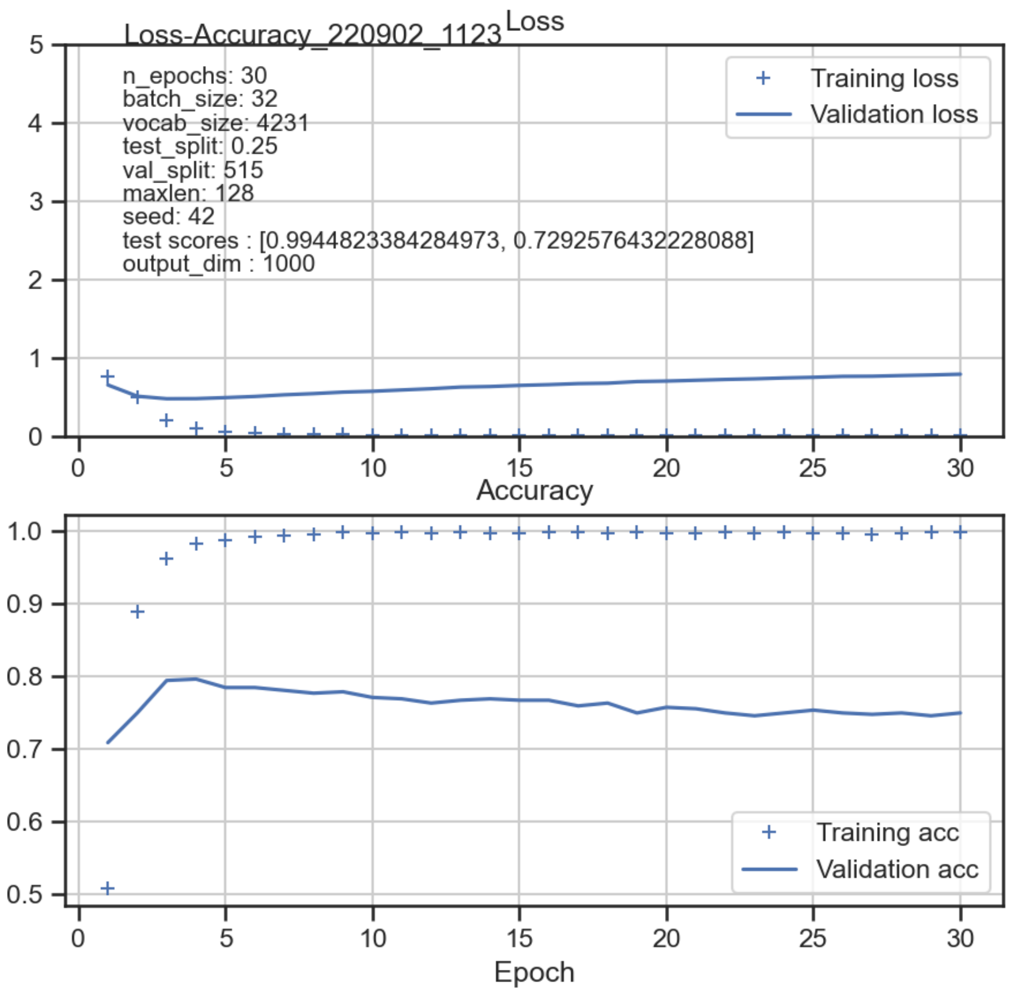
Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 64, 5)	19535
flatten (Flatten)	(None, 320)	0
dense (Dense)	(None, 1)	321

Total params: 19,856
Trainable params: 19,856
Non-trainable params: 0




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 128, 1000)         4231000

 flatten (Flatten)           (None, 128000)            0

 dense (Dense)               (None, 1)                 128001

=================================================================
Total params: 4,359,001
Trainable params: 4,359,001
Non-trainable params: 0
_________________________________________________________________
```

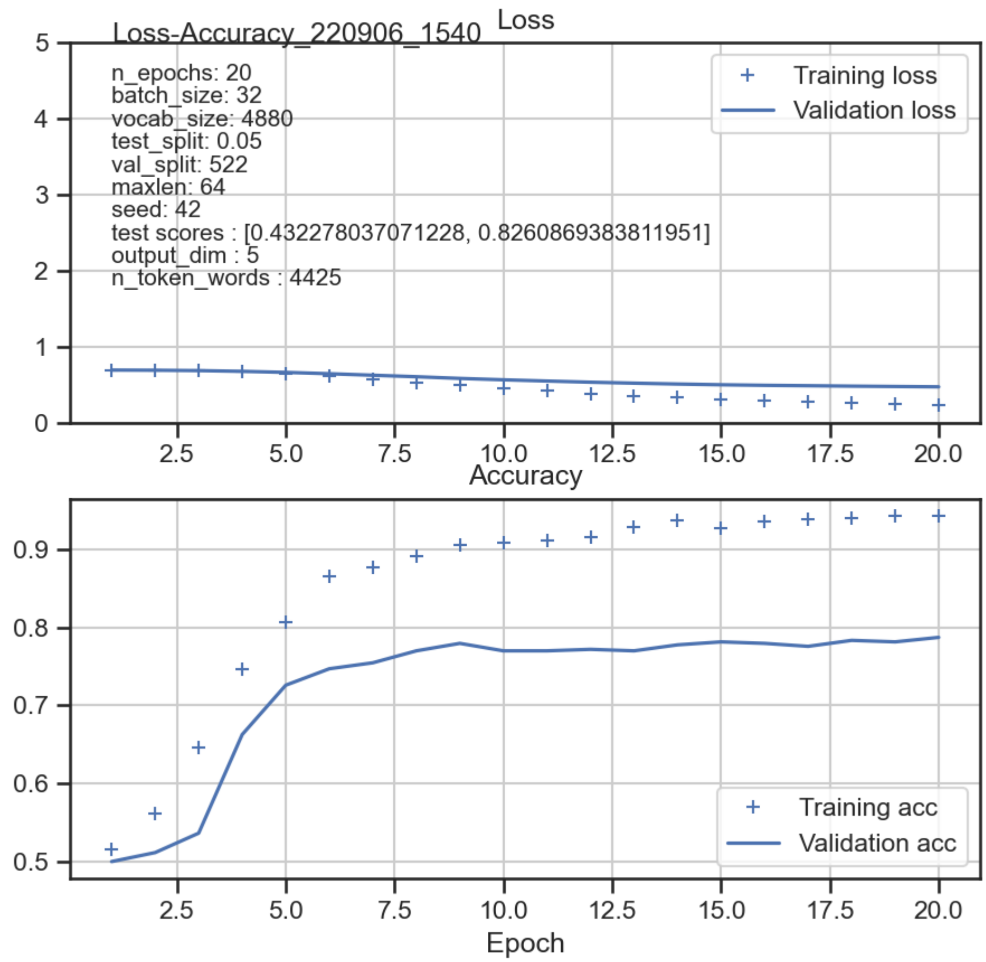



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 128, 1000)         4231000

 flatten (Flatten)           (None, 128000)            0

 dense (Dense)               (None, 1)                 128001

=================================================================
Total params: 4,359,001
Trainable params: 4,359,001
Non-trainable params: 0
_________________________________________________________________
```

Figure 4

Best Model

Best Model




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 64, 5)             24400

 dropout (Dropout)           (None, 64, 5)             0

 flatten (Flatten)           (None, 320)               0

 dense (Dense)               (None, 1)                 321

=================================================================
Total params: 24,721
Trainable params: 24,721
Non-trainable params: 0
_________________________________________________________________
```

Notes. Best model. The best model found included a dropout layer.

Appendix A Python Code

```
#!/usr/bin/env python
# coding: utf-8

# # D213 Task 2 Rev 3 - Mattinson

# ## imports

# In[1]:


# import required libraries
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split as tts
from numpy import array
from keras import models
from keras import layers
from keras import regularizers
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
import wordcloud
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from nltk.stem import WordNetLemmatizer
get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib.pyplot as plt

print('tensorflow ver: {}'.format(tf.__version__))
print('nltk ver: {}'.format(nltk.__version__))
print('wordcloud ver: {}'.format(wordcloud.__version__))
print('numpy ver: {}'.format(np.__version__))
print('pandas ver: {}'.format(pd.__version__))
#print('matplotlib ver: {}'.format(plt.__version__))


# ## get data

# In[2]:


# read csv data
amazon = 'data/amazon_cells_labelled.txt'
imdb =  'data/imdb_labelled.txt'
yelp =  'data/yelp_labelled.txt'
colnames=['text', 'label']
amazon_df = pd.read_csv(amazon, sep='\t', names=colnames, header=None)
imdb_df = pd.read_csv(imdb, sep='\t', names=colnames, header=None)
yelp_df = pd.read_csv(yelp, sep='\t', names=colnames, header=None)
```

```
df = pd.concat([amazon_df, imdb_df, yelp_df])
df = df.reset_index(drop=True)

print('{}\n{}'.format(df.info(), df.shape))
df.sample(5, random_state=0) # 5 random (0) rows of data


# In[3]:


# plot scores as bar plot
print(df['label'].value_counts()) # output to notebook
pd.value_counts(df['label']).plot.bar() # create plot


# In[4]:


# look at 'good' in a negative context
df[(df['text'].str.contains('good') >= 1) & (df['label'] == 0 )]


# In[5]:


# look at 'great' in a negative context
df[(df['text'].str.contains('great') >= 1) & (df['label'] == 0 )]


# In[6]:


# look at 'bad' in a positive context
df[(df['text'].str.contains('bad') >= 1) & (df['label'] == 1 )]


# ## explore data

# In[7]:


# descriptive stattics
print(type(df['label']))
print(df['label'].info())
df.describe()


# ## clean data

# In[8]:


# retype label data


# In[9]:
```

```
# remove punctuation
def remove_punctuation(text: str) -> str:
    '''remove punctuation from text'''
    final = "".join(u for u in text if u not in (
        "?", ".", ";", ":", "!", '"', ','))
    return final # updated string
print('before: {}'.format(df['text'].loc[0]))
df['text'] = df['text'].apply(remove_punctuation)
print('\nafter: {}'.format(df['text'].loc[0]))


# In[10]:


# lower case
print('before: {}'.format(df['text'].loc[0]))
df['text'] = df['text'].astype(str).str.lower()
print('\nafter: {}'.format(df['text'].loc[0]))


# In[11]:


# first tokenization
from nltk.tokenize import RegexpTokenizer
regexp = RegexpTokenizer('\w+')
print('before: {}'.format(df['text'].loc[0]))
df['text_token']=df['text'].apply(regexp.tokenize)
print('\nafter: {}'.format(df['text_token'].loc[0]))

# what is type of the new field
print('\ntext_token type: {}'.format(type(df['text_token'])))


# In[12]:


# remove stopwords
stopwords = nltk.corpus.stopwords.words("english")
print(stopwords[0:20]) # just first 20 stopwords...
#my_stopwords = ['https', 'good', 'great', 'bad']
my_stopwords = ['https']
stopwords.extend(my_stopwords)
print('\nbefore: {}'.format(df['text'].loc[0]))
df['text_token'] = df['text_token'].apply(
    lambda x: [item for item in x if item not in stopwords])
print('\nafter: {}'.format(df['text_token'].loc[0]))


# In[13]:


# remove infrequent words
df['text_string'] = df['text_token'].apply(
    lambda x: ' '.join([item for item in x if len(item)>2]))
all_words = ' '.join([word for word in df['text_string']])
```

```
tokenized_words = nltk.tokenize.word_tokenize(all_words)
from nltk.probability import FreqDist
fdist = FreqDist(tokenized_words)
print(fdist)
cutoff = 1 # drop words occurring less than certain amount
print('\nbefore: {}'.format(df['text'].loc[0]))
df['text_string_fdist'] = df['text_token'].apply(
    lambda x: ' '.join([item for item in x if fdist[item] >= cutoff ]))
print('\nafter (text_string): {}'.format(df['text_string'].loc[0]))
print('\nafter (text_string_fdist):
{}'.format(df['text_string_fdist'].loc[0]))


# In[14]:


# lemmatize
wordnet_lem = WordNetLemmatizer()
print('\nbefore: {}'.format(df['text'].loc[0]))
df['text_string_lem'] = df['text_string_fdist'].apply(wordnet_lem.lemmatize)
print('\nafter (text_string_lem): {}'.format(df['text_string_lem'].loc[0]))


# In[15]:


# Defining our word cloud drawing function
# adapted from Assaker (2022)
# https://github.com/JosephAssaker/Twitter-Sentiment-Analysis-Classical-
Approach-VS-Deep-
Learning/blob/master/Twitter%20Sentiment%20Analysis%20-%20Classical%20Approac
h%20VS%20Deep%20Learning.ipynb
def plot_wordcloud(title: str, data, color = 'black'):
    print(title) # output to notebook
    wordcloud = WordCloud(stopwords = STOPWORDS,
                          background_color = color,
                          width = 2500,
                          height = 2000
                         ).generate(' '.join(data))
    plt.figure(1, figsize = (13, 13))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show() # create output plot


# In[16]:


# finding most common words
n_common = 20
all_words_lem = ' '.join([word for word in df['text_string_lem']])
words = nltk.word_tokenize(all_words_lem)
fd = FreqDist(words)
top_x_words = fd.most_common(n_common)
fdist = pd.Series(dict(top_x_words)) # data converted to series
import seaborn as sns
sns.set_theme(style="ticks")
```

```
sns.barplot(y=fdist.index, x=fdist.values, color='blue');
print(fd.most_common(n_common))


# https://www.kirenz.com/post/2021-12-11-text-mining-and-sentiment-analysis-with-nltk-and-pandas-in-python/text-mining-and-sentiment-analysis-with-nltk-and-pandas-in-python/

# In[17]:


# wordcloud
wordcloud = WordCloud(width=600,
                     height=400,
                     random_state=2,
                     max_font_size=100).generate(all_words_lem)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off');


# ## export clean data

# In[18]:


# review what the data looks like after cleaning
print('{}\n{}'.format(df.info(), df.shape))
df.sample(3, random_state=0) # 5 random (0) rows of data


# In[19]:


# export clean data
f = 'tables\clean.csv'
df.to_csv(f, index=True, header=True)


# ## train test split

# https://www.kaggle.com/code/arunkumarramanan/awesome-ml-and-text-classification-movie-reviews

# ### seed=

# ### test_split=

# In[20]:


# train test split
X = df['text_string_lem']
y = df['label']
seed = 42 # try different seeds
test_split = 0.05 # 0.2 best so far
X_train, X_test, y_train, y_test = tts(X, y,
```

```
        test_size=test_split, random_state=seed)
print(X_train[0:3]) # df['text_string_lem']
print('X_train shape-type: {}-{}'.format(X_train.shape, type(X_train)))
print('X_test shape: {}'.format(X_test.shape))
print('y_train shape-type: {}-{}'.format(y_train.shape, type(y_train)))
print('y_test shape: {}'.format(y_test.shape))


# ## model #1 - keras(Sequential)

# ### n_token_words =

# In[21]:


# second tokenizer words -> numbers
n_token_words = 4425 # best so far = 5000
tokenizer = Tokenizer(num_words=n_token_words)
#print('\ntype: {}\nbefore:\n{}'.format(type(X_test), X_test[0]))
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_sequences(X_train) # ndarry/df -> list
X_test = tokenizer.texts_to_sequences(X_test)
#print('\ntype: {}\nafter:\n{}'.format(type(X_test), X_test[0])) # now a list


# In[22]:


print(type(X_test))


# In[23]:


#X_train[0:3] # tokenized


# ### vocab_size =

# ### maxlen =

# In[24]:


# Adding 1 because of reserved 0 index
vocab_size = len(tokenizer.word_index) + 1
maxlen = 64
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
np.set_printoptions(threshold=np.inf)
print('vocab_size: {}'.format(vocab_size))
print('maxlen: {}'.format(maxlen))
X_test[0] # now a padded list


# In[25]:
```

```
# reset options
#pd.reset_option('all')


# In[26]:


#X_train[0:3] # padded


# ### dropout =

# ### output_dim =

# In[27]:


# define model
dropout = 0.4 # use dropout = 0 to specify not dropout layer
output_dim = 2000 # vocab_size # 1-1 mapping to vocab word
model = models.Sequential()
model.add(layers.Embedding(input_dim=vocab_size, output_dim=output_dim,
input_length=maxlen))
if(dropout > 0):
    model.add(layers.Dropout(dropout))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))

print(model.summary())


# In[28]:


# compile model
model.compile(
            optimizer='adam',
            loss='binary_crossentropy',
            metrics=['acc'])


# In[29]:


# save model in SavedModel format
# prior to saving the model, you need to compile the model
from datetime import datetime
now = datetime.now() # current date and time
date_time_stamp = now.strftime("_%y%m%d_%H%M")
model.save('models/final' + date_time_stamp)


# ### val_split =

# In[30]:
```

```
val_split = .2 # .3 or .4 working best so far
len(X_train)
val_split = int(val_split * len(X_train))
x_val = X_train[:val_split]
partial_x_train = X_train[val_split:]
y_val = y_train[:val_split]
partial_y_train = y_train[val_split:]


# ### batch_size =

# ### n_epochs =

# In[31]:


batch_size = 32 # 256 best so far
n_epochs = 300 # 100-200 best so far
history = model.fit(partial_x_train,
                    partial_y_train,
                    batch_size=batch_size,
                    epochs=n_epochs,
                    verbose=0,
                    validation_data=(x_val, y_val))


# "Usually training should be better than validation..."

# validation loss goes down but then increases - overfit

# ## custom_loss_acc_plot

# In[32]:


import matplotlib.pyplot as plt
import matplotlib.axes as ax
# adapted from Assaker (2022)
def custom_loss_acc_plot(
    ax: ax,
    hist: dict,
    title: str,
    n_epochs: int,
    batch_size: int,
    vocab_size: int,
    output_dim: int,
    test_split: int,
    val_split: int,
    maxlen: int,
    seed: int,
    summary: str,
    top: int,
    score: np.ndarray,
    n_token_words: int,
    dropout: float
) -> ax:
```

```
    """
    custom subplot returns
    """
    # plot loss on axis=0
    y1 = hist['loss']
    y2 = hist['val_loss']
    x = range(1, len(y1) + 1) # x-axis = Epochs
    ax[0].plot(x, y1, 'b+', label='Training loss')
    ax[0].plot(x, y2, 'b', label='Validation loss')
    ax[0].set_title('Loss')
    ax[0].text(.05 * n_epochs, top - .5, 'n_epochs: ' + str(n_epochs),
fontsize=10)
    ax[0].text(.05 * n_epochs, top - .8, 'batch_size: ' + str(batch_size),
fontsize=10)
    ax[0].text(.05 * n_epochs, top - 1.1, 'vocab_size: ' + str(vocab_size),
fontsize=10)
    ax[0].text(.05 * n_epochs, top - 1.4, 'test_split: ' + str(test_split),
fontsize=10)
    ax[0].text(.05 * n_epochs, top - 1.7, 'val_split: ' + str(val_split),
fontsize=10)
    ax[0].text(.05 * n_epochs, top - 2.0, 'maxlen: ' + str(maxlen),
fontsize=10)
    ax[0].text(.05 * n_epochs, top - 2.3, 'seed: ' + str(seed), fontsize=10)
    ax[0].text(.05 * n_epochs, top - 2.6, 'test scores: ' + str(score),
fontsize=10)
    ax[0].text(.05 * n_epochs, top - 2.9, 'output_dim: ' + str(output_dim),
fontsize=10)
    ax[0].text(.05 * n_epochs, top - 3.2, 'n_token_words: ' +
str(n_token_words), fontsize=10)
    if(dropout > 0):
        ax[0].text(.05 * n_epochs, top - 3.5, 'dropout: ' + str(dropout),
fontsize=10)
    ax[0].grid(True)
    ax[0].axis('on')
    ax[0].set_ylim(0,5)
    #ax[0].set_ylim(0,1)
    #ax[0].yaxis.set_major_locator((integer=True))
    ax[0].legend()

    # plot acc on axis=1
    y1 = hist['acc']
    y2 = hist['val_acc']
    x = range(1, len(y1) + 1) # x-axis = Epochs
    ax[1].plot(x, y1, 'b+', label='Training acc')
    ax[1].plot(x, y2, 'b', label='Validation acc')
    ax[1].set_title('Accuracy')
    ax[1].set_xlabel('Epoch')
    ax[1].grid(True)
    ax[1].axis('on')
    ax[1].legend()

    # plot model summary on axis=2
    ax[2].text(0, -.2, summary, fontsize=10)
    ax[2].grid(False)
    ax[2].axis('off')
    return (ax)
```

```
title = 'Loss-Accuracy'
fig, ax = plt.subplots(3, sharex=False, figsize=(7,10))
stringlist = []
model.summary(print_fn=lambda x: stringlist.append(x))
short_model_summary = "\n".join(stringlist)
score = model.evaluate(X_test, y_test, verbose=0)
top = 5
custom_loss_acc_plot(
    ax,
    history.history,
    title,
    n_epochs,
    batch_size,
    vocab_size,
    output_dim,
    test_split,
    val_split,
    maxlen,
    seed,
    short_model_summary,
    top,
    score,
    n_token_words,
    dropout
)

from datetime import datetime
now = datetime.now() # current date and time
title += now.strftime("_%y%m%d_%H%M")
ax[0].text(.05 * n_epochs, top, title, fontsize=12)
fig.savefig('figures\\' + title, dpi=150)
plt.close()


# ## end of notebook

# In[33]:


# beeps to indicate end of notebook
import winsound
n_beeps = int((score[1]*10-5))
for i in range(5):
    winsound.Beep(700, 100)
for i in range(n_beeps):
    winsound.Beep(500, 200)


# In[ ]:
```