Predicting customer product score by review text using LSTM and GRU recurent neural networks

Brief description of the problem and data

The challange problem that will be addressed in this notebook is: given a data set of womens clothing reviews, which among other details includes the text of the review and the customer's score of the product, train a recurrent neural netwrk to predict what score a customer will give a product based on the text of their review.

The data in question is a collection of reviews posted on e-commerce platforms, with each review given to an article of womens clothing. Each sample contains the text of the review itself, the score the customer gave the product, how helpul other customers found the review, and other details as will be explored later.

Data source

The data used in this notebook is from the "Women's E-Commerce Clothing Reviews" dataset, hosted on Kaggle.

URL to data set: https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews

In [70]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import tensorflow as tf
import keras
import keras_nlp

from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix, f1_score, recall_s

import seaborn as sns
import altair as alt
import matplotlib.pyplot as plt

import keras_tuner
import kerastuner
from keras_tuner.tuners import RandomSearch

from sklearn import feature_extraction, linear_model, model_selection, preprocessing

import nltk
from nltk.corpus import stopwords

import json
import os
import sys
import string
import re

os.environ["KERAS_BACKEND"] = "tensorflow"

print("TensorFlow version:", tf.__version__)
```

```
print(f"Keras Version: {keras.__version__}")
print("KerasNLP version:", keras_nlp.__version__)
print()
print(f"Python {sys.version}")
gpu_list = tf.config.list_physical_devices('GPU')
gpu = len(gpu_list)>0
print("GPU is", "available" if gpu else "NOT AVAILABLE")
print(gpu_list)
```

```
TensorFlow version: 2.10.0
Keras Version: 2.10.0
KerasNLP version: 0.6.1

Python 3.9.19 (main, Mar 21 2024, 17:21:27) [MSC v.1916 64 bit (AMD64)]
GPU is available
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Exploratory Data Analysis (EDA) — Inspect, Visualize and Clean the Data

Data inspection

Below is a display of the first 5 samples of the data set.

In [209…
```
path_add = 'C:/Users/kgrit/OneDrive/Documents/UC boulder Data science masters/Intro to d
df_reviews = pd.read_csv(path_add + "data/Womens Clothing E-Commerce Reviews.csv")
```

In [216…
```
print(df_reviews.shape)
display(df_reviews.head())
```

```
(23486, 11)
```

	Unnamed: 0	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Department Name	
0	0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initmates	Intimate	Int
1	1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	Dresses	D
2	2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	Dresses	D
3	3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Petite	Bottoms	
4	4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	Tops	B

The only features relevent to this notebook are 'Review Text' and 'Rating', so those two will be pulled out and placed into a new dataframe. Additionally, rows with no review text or rating will be pulled at the beginning as a preemptive data cleaning operation.

In [ ]:
```
df_reviews_f = df_reviews[['Review Text', 'Rating']].copy().rename(columns={"Review Text
```

In [222…
```
df_reviews_f = df_reviews_f[df_reviews_f['text'].isnull() == False].copy()

display(df_reviews_f.head(n=5))
print('Dataframe shape:', df_reviews_f.shape)
print('rating without text: ',df_reviews_f[df_reviews_f['text'].isnull()].shape[0])
print('Text without rating: ',df_reviews_f[df_reviews_f['rating'].isnull()].shape[0])
```

	text	rating
0	Absolutely wonderful - silky and sexy and comf...	4
1	Love this dress! it's sooo pretty. i happene...	5
2	I had such high hopes for this dress and reall...	3
3	I love, love, love this jumpsuit. it's fun, fl...	5
4	This shirt is very flattering to all due to th...	5

```
Dataframe shape: (22641, 2)
rating without text:  0
Text without rating:  0
```

Of the data samples that have both text and rating, you can see that there are 22,641 samples that can be used in the following attampts. Lets move onto some statistics of the number of ratings for each possibler rating and the text lengths

EDA - statistics and visualizations

First lets take a quick look at the length of the review texts, and some statistics of those lengths.

In [6]:
```
def text_len(row):
    print(row["text"])
    return len(row["text"])
df_reviews_f["text_length"] = df_reviews_f['text'].apply(lambda x : len(x))

print("Full data set Stat\n")
df_reviews_f_len_desc = df_reviews_f["text_length"].describe()
print(df_reviews_f_len_desc)
```

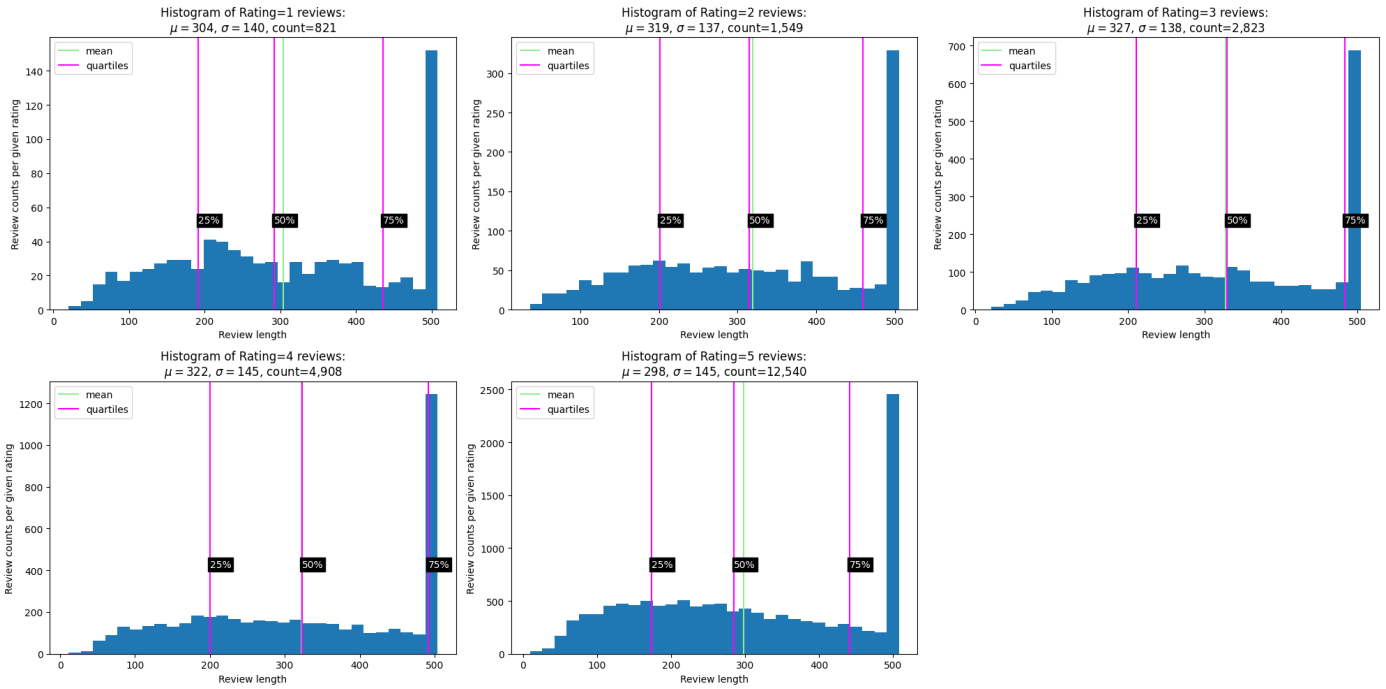
```
Full data set Stat

count    22641.000000
mean       308.687911
std        143.940048
min          9.000000
25%        186.000000
50%        301.000000
75%        459.000000
max        508.000000
Name: text_length, dtype: float64
```

It looks like a large number of reveiw lengths are close to 300 characters, including spaces. However, if you look at the 75% quartile number, you can see that it is quite close to the max possible length. This could

mean that a lot of review lengths tend towards the max length calculated. Lets look at histograms of the review lengths, broken up into 5 charts, one for each possible rating.

In [7]:
```
def plot_histograms(df_in, col='text_length'):
    fig, ax = plt.subplots(2, 3, figsize=(20, 10))
    x = [df_in[df_in['rating'] == i] for i in range(1,6)]
    stats = [df[col].describe() for df in x]

    for i in range(5):
        ax_row = i // 3
        ax_col = i % 3
        n, bins, patches = ax[ax_row, ax_col].hist(x[i][col], 30, density=False)
        mu = stats[i]['mean']
        sigma = stats[i]['std']

        ax[ax_row, ax_col].axvline(mu, color='lightgreen')
        ax[ax_row, ax_col].axvline(stats[i]['25%'], color='magenta')
        ax[ax_row, ax_col].axvline(stats[i]['50%'], color='magenta')
        ax[ax_row, ax_col].axvline(stats[i]['75%'], color='magenta')

        max_c = np.max(n)/3
        if i == -1:
            print(n)
            print(bins)
            #475.46666667
            print(x[i][(x[i][col] < 490) & (x[i][col] > 475)])
            print(x[i][x[i][col] > 490].iloc[0,0])
        percs = ['25%', '50%', '75%']
        for j in range(3):
            ax[ax_row, ax_col].text(stats[i][percs[j]], max_c, percs[j], color='white',
        #ax[i].text(stats[i]['50%'], max_c, '50%', color=t_color)
        #ax[i].text(stats[i]['75%'], max_c, '75%', color=t_color)
        #ax[i].axvline(stats[i]['25%'], color='orange')

        ax[ax_row, ax_col].set_xlabel('Review length')
        ax[ax_row, ax_col].set_ylabel('Review counts per given rating')

        ax[ax_row, ax_col].set_title(f'Histogram of Rating={i+1} reviews:\n'
                    fr'$\mu={mu:.0f}$, $\sigma={sigma:.0f}$, count={"{:,}".format(x[i].s
        ax[ax_row, ax_col].legend(['mean', 'quartiles'])

    # Tweak spacing to prevent clipping of ylabel
    fig.delaxes(ax[1][2])
    fig.tight_layout()
    plt.show()
```

In [8]:
```
plot_histograms(df_reviews_f)
```



As you can see, there are very large numbers of reviews for each possible review rating that are close to 500 characters. It is possible that these reviews have a large number of filler words, or stop words, that greatly exagerate the amount of meaningful words in the longer reviews. That will be where the majority of data cleaning will be focused.

Data cleaning

Data cleaning will be comprised of removeing stop words from review texts, along with removing punctuation to simply text tokenization and make it easier for the RNN models to interpret reviews based on only vocabulary. I will also display the newly leaned review texts lengths in histograms of the same form as above. This will make it clear if the text cleaning smoothed out the distribution of text lengths.

In [11]:
```
'''
def remove_stopword(text):
    remove_stopword = " ".join([word for word in text.split() if word.lower() not in sto

    return remove_stopword

df_reviews_f["text_clean"] = df_reviews_f["text"].apply(remove_stopword)

def clean_text(x):
    return re.sub(r"\s+", " ", x.translate(str.maketrans('', '', string.punctuation))).l

df_reviews_f["text_clean"] = df_reviews_f["text_clean"].apply(lambda x : clean_text(x))

df_reviews_f["length_clean"] = df_reviews_f["text_clean"].apply(lambda x : len(x))

#display(df_train["text_clean"].head())
#display(df_test["text_clean"].head())

df_reviews_f.to_csv("data/Womens Clothing E-Commerce Reviews - clean.csv", index=False)
'''
```

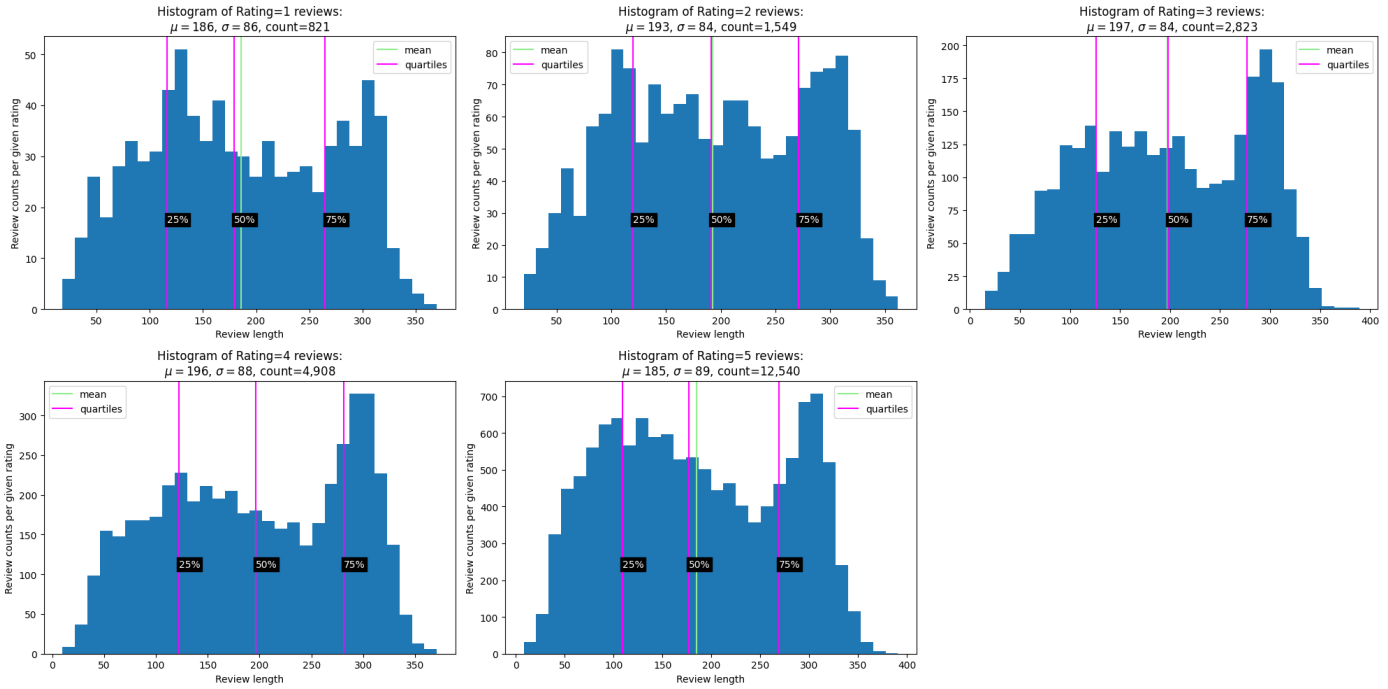
In [13]:
```
df_reviews_f = pd.read_csv("data/Womens Clothing E-Commerce Reviews - clean.csv")
df_reviews_f.head()
```

Out[13]:

	text	rating	text_length	text_clean	length_clean
0	Absolutely wonderful - silky and sexy and comf...	4	53	absolutely wonderful silky sexy comfortable	43
1	Love this dress! it's sooo pretty. i happene...	5	303	love dress sooo pretty happened find store im ...	187
2	I had such high hopes for this dress and reall...	3	500	high hopes dress really wanted work me initial...	312
3	I love, love, love this jumpsuit. it's fun, fl...	5	124	love love love jumpsuit fun flirty fabulous ev...	92
4	This shirt is very flattering to all due to th...	5	192	shirt flattering due adjustable front tie perf...	112

Lets look at histogram charts again to view the text lengths distribution for each possible product rating

In [14]:
```
plot_histograms(df_reviews_f, col='length_clean')
```



The text cleaning was able to smooth out the distributions quite a bit. Also, as you can see by looking at the spread of review length counts, the text cleaning brought every set of review texts into a similar spread of possible review lengths.

It appears that in almost every histogram there are two overlapping gaussian peaks in the data, one with a mean of about 300 and another with a mean of around 150. This could be because of the set of reviews with close to 500 characters. These reviews could have had proportional amounts of meaningful vocabulary in them when compared to the shorter reviews. We will keep all the reviews in one data set regardless. Future attempts to use the review texts here to predict review ratings could break the two sets of reivew lengths appeart and run model fitting on each one seperatly, or some other process.

Data preprocessing and transformation

Splitting into testing, traing, and validation data

Lets set up the data for use in the model fittings. I will split the collections of each rating, then combign and shuffle them into the training, validation and testing data sets. This will be done using the test_train_split

function. I'll then use the Word2Vec approach to preprocess the data and utilize the keras tokenizer() function to convert the review texts to a matrix of integers.

In [20]:
```
df_reviews_f["rating_shifted"] = df_reviews_f['rating'].apply(lambda x: x - 1).values

df_train, df_test = train_test_split(
    df_reviews_f,
    test_size=0.2,
    random_state=42,
    stratify=df_reviews_f['rating'].values,)

df_val = df_train.sample(frac=0.3)
df_train.drop(df_val.index, inplace=True)
```

In [223…
```
def df_stats_print(df, df_title):
    print('\n' + df_title)
    #display(df.describe())
    for i in range(1, 6):
        _count = df[df['rating'] == i]['rating'].count()
        _total = df.shape[0]
        print(f"{i} counts: {_count}, % of total: {np.round(_count/_total*100,2)}")

df_stats_print(df_train, 'df_train')
df_stats_print(df_test, 'df_test')
df_stats_print(df_val, 'df_val')
#display(df_train.head())
```

```
df_train
1 counts: 479, % of total: 3.78
2 counts: 865, % of total: 6.82
3 counts: 1598, % of total: 12.6
4 counts: 2737, % of total: 21.59
5 counts: 6999, % of total: 55.21

df_test
1 counts: 164, % of total: 3.62
2 counts: 310, % of total: 6.84
3 counts: 565, % of total: 12.48
4 counts: 982, % of total: 21.68
5 counts: 2508, % of total: 55.38

df_val
1 counts: 178, % of total: 3.28
2 counts: 374, % of total: 6.88
3 counts: 660, % of total: 12.15
4 counts: 1189, % of total: 21.88
5 counts: 3033, % of total: 55.82
```

Above you can see that each possible rating value is proportionally represented in the training, validation, and testing data sets.

In [22]:
```
max_features=5000
tokenizer=keras.preprocessing.text.Tokenizer(num_words=max_features,split=' ')
tokenizer.fit_on_texts(df_train['text_clean'].values)

X_train = tokenizer.texts_to_sequences(df_train['text_clean'].values)
X_train = keras.utils.pad_sequences(X_train)

X_val = tokenizer.texts_to_sequences(df_val['text_clean'].values)
X_val = keras.utils.pad_sequences(X_val, maxlen=X_train.shape[1])

X_test = tokenizer.texts_to_sequences(df_test['text_clean'].values)
X_test = keras.utils.pad_sequences(X_test,maxlen=X_train.shape[1])
```

To be able to pass the review ratings to the models, they must be formatted into arrays of shape (# of total labels) x (# of total samples) and the reviews need to be shifted down by 1 so that they start at zero. This is necessary for the evalutaion mettcs in the model fitting process to accuratly read the true labels. Each column of the new array will correspond to a given label by order, such that the first column corresponds to rating = 1, the second column to rating = 2, and so on.

In [168…
```
y_train = np.array([ [1 if row == i else 0 for i in range(5)] for row in df_train["ratin

y_val = np.array([ [1 if row == i else 0 for i in range(5)] for row in df_val["rating_sh

y_test = np.array([ [1 if row == i else 0 for i in range(5)] for row in df_test["rating_
#X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [226…
```
print(df_train['rating'].values[:5])
print(y_train[:5])
```

```
[5 1 5 4 5]
[[0 0 0 0 1]
 [1 0 0 0 0]
 [0 0 0 0 1]
 [0 0 0 1 0]
 [0 0 0 0 1]]
```

Below is a test of the tokenizer. The dictionary relating a column index to a specific word is retrived from the tokenizer and used to verify a given row of the formatted training data can be reconstructed (roughly) to match the original cleaned tweet.

In [24]:
```
test_row = 20
print(X_train.shape)
print(X_val.shape)
print(y_train.shape)
print(y_val.shape)
print('test:\n    ', tokenizer.sequences_to_texts([X_train[test_row]]))
print('original cleaned tweet:\n    ', df_train.iloc[test_row]['text_clean'])
```

```
(12678, 61)
(5434, 61)
(12678, 5)
(5434, 5)
test:
     ['made wedding outfit perfect']
original cleaned tweet:
     made wedding outfit perfect
```

Looks good.

Hyperparameter tuning

Every model architecture will have the LSTM/GRU layer units, Dropout layer dropout rates, and learning rates tuned such that the combination of parameters that produces the model with the highest AUC score will be selected as best. Each layer, be it LSTM, GRU or dropout, will have a unique hyperparameter such that it is possible for every dropout layer to have a different dropout rate and every LSTM/GRU layer to have a different number of memory units. One model, the 1-layer GRU model, will have a larger set of possible memory unit hyperparameters, to test if the one GRU layer with more memory units can outperform the other architectures. The list below shows the values tested:

- LSTM/GRU layer memory units: [8, 16,32,64]

- 1-layer GRU layer memory units: [8, 16, 32, 64, 128, 256, 512]
- Dropout layer dropout rates: [0.0, 0.1, 0.2, 0.3, 0.4]
- Learning rates: [0.01, 0.001, 0.0001, 0.00001]

Model architectures

Long-Short Term Memory (LSTM) architecture

This will be a 2-layer LSTM model. To help address overfitting, dropout layers are added directly before, between, after each layer. The metric to maximize in training is multi-label AUC average. The reasoning to use AUC is that since these reviews are being judged as being any one of 5 different possible scores, and to keep the model from over-prioritizing one review rating label right, you can score a model by getting the AUC for each label, then averaging the scores.

In [34]:
```
def lstm_model(hp):

    poss_units = [8,16,32,64, 128]
    hp_ltsm_units_1 = hp.Choice("ltsm_memory_units_layer_1", poss_units)
    hp_ltsm_units_2 = hp.Choice("ltsm_memory_units_layer_2", poss_units)

    poss_dropouts = [0.0, 0.1, 0.2, 0.3, 0.4]
    hp_dropout_frac_1 = hp.Choice("dropout_rate_layer_1", poss_dropouts)
    hp_dropout_frac_2 = hp.Choice("dropout_rate_layer_2", poss_dropouts)
    hp_dropout_frac_3 = hp.Choice("dropout_rate_layer_3", poss_dropouts)

    embed_dim = 100

    model = keras.models.Sequential([
        keras.layers.Embedding(max_features, embed_dim, input_length = X_train.shape[1])
        keras.layers.Dropout(hp_dropout_frac_1),
        keras.layers.LSTM(hp_ltsm_units_1, return_sequences=True, name="LSTM_layer_1"),
        keras.layers.Dropout(hp_dropout_frac_2),
        keras.layers.LSTM(hp_ltsm_units_2, name="LSTM_layer_2"),
        keras.layers.Dropout(hp_dropout_frac_3),
        keras.layers.Dense(5,activation='softmax'),
        ])

    hp_lr = hp.Choice("learning_rate", [1e-2, 1e-3, 1e-4, 1e-5])

    model.compile(
        loss=keras.losses.CategoricalCrossentropy(),
        optimizer=keras.optimizers.Adam(learning_rate=hp_lr),
        metrics= [keras.metrics.AUC(multi_label=True, num_labels=5, name='auc', from_log
    )

    return model
```

GRU 1-layer architecture

This will be a single layer GRU model. To help address overfitting, dropout layers are added directly before and after. The metric to maximize in training is multi-label AUC average

In [194…
```
def gru_1_layer_model(hp):

    embed_dim = 100

    poss_units = [8,16,32,64,128,256,512]
```

```
    hp_gru_units_1 = hp.Choice("gru_memory_units_layer_1", poss_units)

    poss_dropouts = [0.0, 0.1, 0.2, 0.3, 0.4]
    hp_dropout_frac_1 = hp.Choice("dropout_rate_layer_1", poss_dropouts)
    hp_dropout_frac_2 = hp.Choice("dropout_rate_layer_2", poss_dropouts)

    model = keras.models.Sequential()
    model.add(keras.layers.Embedding(max_features, embed_dim, input_length = X_train.sha
    model.add(keras.layers.Dropout(hp_dropout_frac_1))
    model.add(tf.keras.layers.GRU(hp_gru_units_1))
    model.add(keras.layers.Dropout(hp_dropout_frac_2))
    model.add(keras.layers.Dense(5,activation='softmax'))


    hp_lr = hp.Choice("learning_rate", [1e-2, 1e-3, 1e-4, 1e-5])

    model.compile(
        loss=keras.losses.CategoricalCrossentropy(),
        optimizer=keras.optimizers.Adam(learning_rate=hp_lr),
        metrics= [keras.metrics.AUC(multi_label=True, num_labels=5, name='auc', from_log
    )

    return model
```

GRU 2-layer architecture

This will be a two layer GRU model. To help address overfitting, dropout layers are added directly before and after. The metric to maximize in training is multi-label AUC average

In [155…

```
def gru_model(hp):

    embed_dim = 100

    poss_units = [8,16,32,64]
    hp_gru_units_1 = hp.Choice("gru_memory_units_layer_1", poss_units)
    hp_gru_units_2 = hp.Choice("gru_memory_units_layer_2", poss_units)

    poss_dropouts = [0.0, 0.1, 0.2, 0.3, 0.4]
    hp_dropout_frac_1 = hp.Choice("dropout_rate_layer_1", poss_dropouts)
    hp_dropout_frac_2 = hp.Choice("dropout_rate_layer_2", poss_dropouts)
    hp_dropout_frac_3 = hp.Choice("dropout_rate_layer_3", poss_dropouts)

    model = keras.models.Sequential()
    model.add(keras.layers.Embedding(max_features, embed_dim, input_length = X_train.sha
    model.add(keras.layers.Dropout(hp_dropout_frac_1))
    model.add(tf.keras.layers.GRU(hp_gru_units_1, return_sequences=True, ))
    model.add(keras.layers.Dropout(hp_dropout_frac_2))
    model.add(tf.keras.layers.GRU(hp_gru_units_2))
    model.add(keras.layers.Dropout(hp_dropout_frac_3))
    model.add(keras.layers.Dense(5,activation='softmax'))


    hp_lr = hp.Choice("learning_rate", [1e-2, 1e-3, 1e-4, 1e-5])

    model.compile(
        loss=keras.losses.CategoricalCrossentropy(),
        optimizer=keras.optimizers.Adam(learning_rate=hp_lr),
        metrics= [keras.metrics.AUC(multi_label=True, num_labels=5, name='auc', from_log
    )

    return model
```

GRU 4 layer

This will be a four layer GRU model. To help address overfitting, dropout layers are added directly before and after. The metric to maximize in training is multi-label AUC average

In [29]:
```
def gru_4_layer_model(hp):

    embed_dim = 100
    4*4*5*5*4
    poss_units = [8,16,32,64]
    hp_gru_units_1 = hp.Choice("gru_memory_units_layer_1", poss_units)
    hp_gru_units_2 = hp.Choice("gru_memory_units_layer_2", poss_units)
    hp_gru_units_3 = hp.Choice("gru_memory_units_layer_3", poss_units)
    hp_gru_units_4 = hp.Choice("gru_memory_units_layer_4", poss_units)

    poss_dropouts = [0.0, 0.1, 0.2, 0.3, 0.4]
    hp_dropout_frac_1 = hp.Choice("dropout_rate_layer_1", poss_dropouts)
    hp_dropout_frac_2 = hp.Choice("dropout_rate_layer_2", poss_dropouts)
    hp_dropout_frac_3 = hp.Choice("dropout_rate_layer_3", poss_dropouts)
    hp_dropout_frac_4 = hp.Choice("dropout_rate_layer_4", poss_dropouts)
    hp_dropout_frac_5 = hp.Choice("dropout_rate_layer_5", poss_dropouts)

    model = keras.models.Sequential()
    model.add(keras.layers.Embedding(max_features, embed_dim, input_length = X_train.sha
    model.add(keras.layers.Dropout(hp_dropout_frac_1))
    model.add(tf.keras.layers.GRU(hp_gru_units_1, return_sequences=True))
    model.add(keras.layers.Dropout(hp_dropout_frac_2))
    model.add(tf.keras.layers.GRU(hp_gru_units_2, return_sequences=True))
    model.add(keras.layers.Dropout(hp_dropout_frac_3))
    model.add(tf.keras.layers.GRU(hp_gru_units_3, return_sequences=True))
    model.add(keras.layers.Dropout(hp_dropout_frac_4))
    model.add(tf.keras.layers.GRU(hp_gru_units_4))
    model.add(keras.layers.Dropout(hp_dropout_frac_5))
    model.add(keras.layers.Dense(5,activation='softmax'))


    hp_lr = hp.Choice("learning_rate", [1e-2, 1e-3, 1e-4, 1e-5])

    model.compile(
        loss=keras.losses.CategoricalCrossentropy(),
        optimizer=keras.optimizers.Adam(learning_rate=hp_lr),
        metrics= [keras.metrics.AUC(multi_label=True, num_labels=5, name='auc', from_log
    )

    return model
```

Tuners to find optimal hyperparametrs

Below are the different hyperparameter tuners for each of the models defined above. Each tuner will randomly select 50 different sets of possible hyperparameters for each model, train each of the 50 models for 3 epochs, then sgrade these 50 models based on total AUC score.

In [32]:
```
tuner_files_dir = 'C:/Users/kgrit/OneDrive/Documents/UC boulder Data science masters/Int
t_objective = kerastuner.Objective('auc', direction='max')
t_max_trials = 50
t_overwrite = False
t_executions_per_trial = 1
```

LSTM tuner

In [36]:
```
tuner_lstm = RandomSearch(
    lstm_model,
    objective=t_objective,
    max_trials=t_max_trials,
    overwrite=t_overwrite,
    executions_per_trial=t_executions_per_trial,
    directory=tuner_files_dir,
    project_name='lstm_model')
```

```
Reloading Tuner from C:/Users/kgrit/OneDrive/Documents/UC boulder Data science masters/Intro to deep learning/final project/rough draft\lstm_model\tuner0.json
```

GRU 1-layer tuner

In [195…
```
tuner_gru_1_layer = RandomSearch(
    gru_1_layer_model,
    objective=t_objective,
    max_trials=t_max_trials,
    overwrite=t_overwrite,
    executions_per_trial=t_executions_per_trial,
    directory=tuner_files_dir,
    project_name='gru_1_layer_model')
```

GRU 2-layer tuner

In [156…
```
tuner_gru = RandomSearch(
    gru_model,
    objective=t_objective,
    max_trials=t_max_trials,
    overwrite=t_overwrite,
    executions_per_trial=t_executions_per_trial,
    directory=tuner_files_dir,
    project_name='gru_model')
```

GRU 4-layer tuner

In [41]:
```
tuner_gru_4_L = RandomSearch(
    gru_4_layer_model,
    objective=t_objective,
    max_trials=t_max_trials,
    overwrite=t_overwrite,
    executions_per_trial=t_executions_per_trial,
    directory=tuner_files_dir,
    project_name='gru_4_layer_model')
```

```
Reloading Tuner from C:/Users/kgrit/OneDrive/Documents/UC boulder Data science masters/Intro to deep learning/final project/rough draft\gru_4_layer_model\tuner0.json
```

Results and Analysis

Now lets run the tuners and find the best set of hyperparameters for each model architecture. Below is code to retrive parameters from previous trials.

LSTM 2 layer tuner results

In [525…
```
tuner_lstm.search(X_train, y_train, epochs=3, validation_data=(X_val, y_val))
```

```
Trial 50 Complete [00h 00m 19s]
auc: 0.5106742978096008
```

```
Best auc So Far: 0.8990048170089722
Total elapsed time: 00h 22m 13s
```

In [39]:
```
models_lstm = tuner_lstm.get_best_models(num_models=5)
best_model_lstm = models_lstm[0]
best_hps_lstm = tuner_lstm.get_best_hyperparameters(1)[0]
print(best_hps_lstm.values)
best_model_lstm.summary()
```

```
{'ltsm_memory_units_layer_1': 64, 'ltsm_memory_units_layer_2': 128, 'dropout_rate_layer_
1': 0.1, 'dropout_rate_layer_2': 0.3, 'dropout_rate_layer_3': 0.1, 'learning_rate': 0.00
1}
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 61, 100)           500000

 dropout (Dropout)           (None, 61, 100)           0

 LSTM_layer_1 (LSTM)         (None, 61, 64)            42240

 dropout_1 (Dropout)         (None, 61, 64)            0

 LSTM_layer_2 (LSTM)         (None, 128)               98816

 dropout_2 (Dropout)         (None, 128)               0

 dense (Dense)               (None, 5)                 645

=================================================================
Total params: 641,701
Trainable params: 641,701
Non-trainable params: 0
_________________________________________________________________
```

GRU 1 layer tuner results

In [196…
```
with tf.device('/GPU:0'):
    tuner_gru_1_layer.search(X_train, y_train, epochs=3, validation_data=(X_val, y_val))
```

```
Trial 50 Complete [00h 00m 13s]
auc: 0.6010494232177734

Best auc So Far: 0.9196551442146301
Total elapsed time: 00h 13m 55s
```

In [197…
```
models_gru_1_layer = tuner_gru_1_layer.get_best_models(num_models=5)
best_model_gru_1_layer = models_gru_1_layer[0]
best_hps_gru_1_layer = tuner_gru_1_layer.get_best_hyperparameters(1)[0]
print(best_hps_gru_1_layer.values)
best_model_gru_1_layer.summary()
```

```
{'gru_memory_units_layer_1': 256, 'dropout_rate_layer_1': 0.3, 'dropout_rate_layer_2':
0.0, 'learning_rate': 0.01}
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 61, 100)           500000

 dropout (Dropout)           (None, 61, 100)           0

 gru (GRU)                   (None, 256)               274944

 dropout_1 (Dropout)         (None, 256)               0
```

```
 dense (Dense)               (None, 5)                 1285

=================================================================
Total params: 776,229
Trainable params: 776,229
Non-trainable params: 0
_________________________________________________________________
```

GRU 2 layer tuner results

In [158…]:
```
with tf.device('/GPU:0'):
    tuner_gru.search(X_train, y_train, epochs=3, validation_data=(X_val, y_val))
```

```
Trial 50 Complete [00h 00m 29s]
auc: 0.5173693895339966

Best auc So Far: 0.9008854627609253
Total elapsed time: 00h 17m 44s
```

In [159…]:
```
models_gru = tuner_gru.get_best_models(num_models=5)
best_model_gru = models_gru[0]
best_hps_gru = tuner_gru.get_best_hyperparameters(1)[0]
print(best_hps_gru.values)
best_model_gru.summary()
```

```
{'gru_memory_units_layer_1': 8, 'gru_memory_units_layer_2': 16, 'dropout_rate_layer_1':
0.1, 'dropout_rate_layer_2': 0.0, 'dropout_rate_layer_3': 0.2, 'learning_rate': 0.01}
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 61, 100)           500000

 dropout (Dropout)           (None, 61, 100)           0

 gru (GRU)                   (None, 61, 8)             2640

 dropout_1 (Dropout)         (None, 61, 8)             0

 gru_1 (GRU)                 (None, 16)                1248

 dropout_2 (Dropout)         (None, 16)                0

 dense (Dense)               (None, 5)                 85

=================================================================
Total params: 503,973
Trainable params: 503,973
Non-trainable params: 0
_________________________________________________________________
```

GRU 4 layer tuner results

In [31]:
```
tuner_gru_4_L.search(X_train, y_train, epochs=3, validation_data=(X_val, y_val))
```

```
Trial 50 Complete [00h 00m 31s]
auc: 0.5377457737922668

Best auc So Far: 0.8823961019515991
Total elapsed time: 00h 39m 30s
```

In [42]:
```
models_gru_4_L = tuner_gru_4_L.get_best_models(num_models=5)
best_model_gru_4_L = models_gru_4_L[0]
best_hps_gru_4_L = tuner_gru_4_L.get_best_hyperparameters(1)[0]
```

```
print(best_hps_gru_4_L.values)
best_model_gru_4_L.summary()
```

```
{'gru_memory_units_layer_1': 8, 'gru_memory_units_layer_2': 32, 'gru_memory_units_layer_
3': 32, 'gru_memory_units_layer_4': 32, 'dropout_rate_layer_1': 0.3, 'dropout_rate_layer
_2': 0.0, 'dropout_rate_layer_3': 0.0, 'dropout_rate_layer_4': 0.0, 'dropout_rate_layer_
5': 0.2, 'learning_rate': 0.01}
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 61, 100)           500000

 dropout (Dropout)           (None, 61, 100)           0

 gru (GRU)                   (None, 61, 8)             2640

 dropout_1 (Dropout)         (None, 61, 8)             0

 gru_1 (GRU)                 (None, 61, 32)            4032

 dropout_2 (Dropout)         (None, 61, 32)            0

 gru_2 (GRU)                 (None, 61, 32)            6336

 dropout_3 (Dropout)         (None, 61, 32)            0

 gru_3 (GRU)                 (None, 32)                6336

 dropout_4 (Dropout)         (None, 32)                0

 dense (Dense)               (None, 5)                 165

=================================================================
Total params: 519,509
Trainable params: 519,509
Non-trainable params: 0
_________________________________________________________________
```

Below are reprints of the optimal prameters found for each model architecture

In [231…

```
print('\nLSTM 2 layer tuner results')
display(best_hps_lstm.values)
print('\nGRU 1 layer tuner results')
display(best_hps_gru_1_layer.values)
print('\nGRU 2 layer tuner results')
display(best_hps_gru.values)
print('\nGRU 4 layer tuner results')
display(best_hps_gru_4_L.values)
```

```
LSTM 2 layer tuner results
{'ltsm_memory_units_layer_1': 64,
 'ltsm_memory_units_layer_2': 128,
 'dropout_rate_layer_1': 0.1,
 'dropout_rate_layer_2': 0.3,
 'dropout_rate_layer_3': 0.1,
 'learning_rate': 0.001}
GRU 1 layer tuner results
{'gru_memory_units_layer_1': 256,
 'dropout_rate_layer_1': 0.3,
 'dropout_rate_layer_2': 0.0,
 'learning_rate': 0.01}
GRU 2 layer tuner results
{'gru_memory_units_layer_1': 8,
 'gru_memory_units_layer_2': 16,
 'dropout_rate_layer_1': 0.1,
```

```
 'dropout_rate_layer_2': 0.0,
 'dropout_rate_layer_3': 0.2,
 'learning_rate': 0.01}
GRU 4 layer tuner results
{'gru_memory_units_layer_1': 8,
 'gru_memory_units_layer_2': 32,
 'gru_memory_units_layer_3': 32,
 'gru_memory_units_layer_4': 32,
 'dropout_rate_layer_1': 0.3,
 'dropout_rate_layer_2': 0.0,
 'dropout_rate_layer_3': 0.0,
 'dropout_rate_layer_4': 0.0,
 'dropout_rate_layer_5': 0.2,
 'learning_rate': 0.01}
```

Best model fitting for LTSM, GRU, and 4-layer GRU models

Here the haperparameters from the best models found earlier will be used to construct and train new models for the hyperparameters given architecture. To guard against overfitting, the models will have a callback that will stop fitting when the validation loss increases in value continuously over two epochs. If that is detected and the fitting is stopped, the weights of the best performing (lowest loss) epoch will be restored.

In [48]:
```
batch_size = 32
epochs = 15
early_stopping_p = 5
```

In [47]:
```
callbacks_lstm = [
    keras.callbacks.ModelCheckpoint(filepath="final_lstm_model\model_at_epoch_{epoch}.ke
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=early_stopping_p, restore
]

final_lstm_model = lstm_model(best_hps_lstm)

final_lstm_model_history = final_lstm_model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    batch_size=batch_size,
    epochs=epochs,
    callbacks=callbacks_lstm,
)
```

```
Epoch 1/15
397/397 [==============================] - 7s 14ms/step - loss: 1.0006 - auc: 0.7776 - v
al_loss: 0.9077 - val_auc: 0.8334
Epoch 2/15
397/397 [==============================] - 5s 13ms/step - loss: 0.8050 - auc: 0.8664 - v
al_loss: 0.8838 - val_auc: 0.8375
Epoch 3/15
397/397 [==============================] - 5s 13ms/step - loss: 0.7124 - auc: 0.8978 - v
al_loss: 0.9173 - val_auc: 0.8260
Epoch 4/15
397/397 [==============================] - 5s 13ms/step - loss: 0.6303 - auc: 0.9193 - v
al_loss: 1.0262 - val_auc: 0.8148
Epoch 5/15
397/397 [==============================] - 5s 13ms/step - loss: 0.5539 - auc: 0.9367 - v
al_loss: 1.0731 - val_auc: 0.8048
Epoch 6/15
397/397 [==============================] - 5s 13ms/step - loss: 0.4837 - auc: 0.9502 - v
al_loss: 1.1385 - val_auc: 0.7970
Epoch 7/15
397/397 [==============================] - 5s 14ms/step - loss: 0.4320 - auc: 0.9602 - v
al_loss: 1.2717 - val_auc: 0.7847
```

In [232…
```
final_lstm_model.save("final_lstm_model/final_lstm_model.keras")
```

In [198…
```
callbacks_gru_1_layer = [
    keras.callbacks.ModelCheckpoint(filepath="final_gru_1_layer_model\model_at_epoch_{ep
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=early_stopping_p, restore
]

final_gru_1_layer_model = gru_1_layer_model(best_hps_gru_1_layer)

final_gru_1_layer_model_history = final_gru_1_layer_model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    batch_size=batch_size,
    epochs=epochs,
    callbacks=callbacks_gru_1_layer,
)
```

```
Epoch 1/15
397/397 [==============================] - 5s 9ms/step - loss: 0.9966 - auc: 0.7848 - va
l_loss: 0.8905 - val_auc: 0.8379
Epoch 2/15
397/397 [==============================] - 3s 8ms/step - loss: 0.8167 - auc: 0.8647 - va
l_loss: 0.9227 - val_auc: 0.8213
Epoch 3/15
397/397 [==============================] - 4s 10ms/step - loss: 0.7818 - auc: 0.8798 - v
al_loss: 1.0584 - val_auc: 0.7909
Epoch 4/15
397/397 [==============================] - 4s 10ms/step - loss: 0.8893 - auc: 0.8485 - v
al_loss: 1.7444 - val_auc: 0.6132
Epoch 5/15
397/397 [==============================] - 4s 9ms/step - loss: 1.3606 - auc: 0.6695 - va
l_loss: 1.3322 - val_auc: 0.6749
Epoch 6/15
397/397 [==============================] - 4s 9ms/step - loss: 1.2331 - auc: 0.6973 - va
l_loss: 1.2754 - val_auc: 0.6749
```

In [233…
```
final_gru_1_layer_model.save("final_gru_model/final_gru_1_layer_model.keras")
```

In [160…
```
callbacks_gru = [
    keras.callbacks.ModelCheckpoint(filepath="final_gru_model\model_at_epoch_{epoch}.ker
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=early_stopping_p, restore
]

final_gru_model = gru_model(best_hps_gru)

final_gru_model_history = final_gru_model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    batch_size=batch_size,
    epochs=epochs,
    callbacks=callbacks_gru,
)
```

```
Epoch 1/15
397/397 [==============================] - 18s 42ms/step - loss: 0.9854 - auc: 0.7876 -
val_loss: 0.8944 - val_auc: 0.8317
Epoch 2/15
397/397 [==============================] - 6s 14ms/step - loss: 0.7953 - auc: 0.8709 - v
al_loss: 0.9058 - val_auc: 0.8308
Epoch 3/15
397/397 [==============================] - 4s 10ms/step - loss: 0.6893 - auc: 0.9039 - v
al_loss: 0.9629 - val_auc: 0.8184
```

```
Epoch 4/15
397/397 [==============================] - 4s 10ms/step - loss: 0.6119 - auc: 0.9238 - v
al_loss: 1.0403 - val_auc: 0.8073
Epoch 5/15
397/397 [==============================] - 4s 10ms/step - loss: 0.5742 - auc: 0.9334 - v
al_loss: 1.1154 - val_auc: 0.7932
Epoch 6/15
397/397 [==============================] - 4s 10ms/step - loss: 0.5319 - auc: 0.9424 - v
al_loss: 1.1703 - val_auc: 0.7807
```

In [234…]
```
final_gru_model.save("final_gru_model/final_gru_2_layer_model.keras")
```

In [51]:
```
callbacks_gru_4_L = [
    keras.callbacks.ModelCheckpoint(filepath="final_gru_4_l_model\model_at_epoch_{epoch}
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=early_stopping_p, restore
]

final_gru_4_L_model = gru_4_layer_model(best_hps_gru_4_L)

final_gru_model_history = final_gru_4_L_model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    batch_size=batch_size,
    epochs=epochs,
    callbacks=callbacks_gru_4_L,
)
```

```
Epoch 1/15
397/397 [==============================] - 31s 71ms/step - loss: 1.0049 - auc: 0.7728 -
val_loss: 0.9036 - val_auc: 0.8298
Epoch 2/15
397/397 [==============================] - 27s 68ms/step - loss: 0.8301 - auc: 0.8554 -
val_loss: 0.9475 - val_auc: 0.8212
Epoch 3/15
397/397 [==============================] - 8s 21ms/step - loss: 0.7469 - auc: 0.8857 - v
al_loss: 0.9469 - val_auc: 0.8170
Epoch 4/15
397/397 [==============================] - 7s 18ms/step - loss: 0.7257 - auc: 0.8916 - v
al_loss: 1.0494 - val_auc: 0.7979
Epoch 5/15
397/397 [==============================] - 7s 18ms/step - loss: 0.7347 - auc: 0.8894 - v
al_loss: 1.0631 - val_auc: 0.7975
Epoch 6/15
397/397 [==============================] - 7s 18ms/step - loss: 0.7678 - auc: 0.8783 - v
al_loss: 1.0410 - val_auc: 0.7932
```

In [235…]
```
final_gru_4_L_model.save("final_gru_4_l_model/final_gru_4_L_model.keras")
```

Results and Analysis (35 pts)

Hyperparameter optimization procedure summary

As stated previously, the main layers unit number, the dense layer drop out rate, and the learning rates were all tunedto find the best performing models for each of the three architectures.

The procedure was simply:

1. Generate the list of all combinations of hyperparameters to test and what metric to judge the models by using keras_tuner

2. Train one model per each hyperparamter set for 5 epochs
3. Choose the model + hyperparamter set that achieved the highest recall score

Final model confusion matrices

Below are confusion matrices that showcase how each final model did when predicting the rating given with a particular review text, for both training and validation data set predictions.

In [250…

```
def get_labels(vals_in):
        list_out = [0 for i in vals_in]
        for i in range(len(vals_in)):
            list_out[i] = np.argmax(vals_in[i]) + 1
        return list_out

def displayConfusionMatrix(y_true_train, y_pred_train, model=None, data_set=None, y_true



    num_cols = 1
    if y_true_val is not None:
        num_cols = 2
        y_true_val_f = get_labels(y_true_val)
        y_pred_val_f = get_labels(y_pred_val)

    fig, ax = plt.subplots(1,num_cols, figsize=(10, 4))

    y_true_train_f = get_labels(y_true_train)
    y_pred_train_f = get_labels(y_pred_train)

    ax_in = None
    if y_true_val is not None:
        ax_in = ax[0]
    else:
        ax_in = ax

    disp_1 = ConfusionMatrixDisplay.from_predictions(
        y_true_train_f,
        y_pred_train_f,
        normalize='true',
        values_format=".2f",
        ax=ax_in, cmap=plt.cm.Blues
    )

    if y_true_val is not None:
        disp_2 = ConfusionMatrixDisplay.from_predictions(
            y_true_val_f,
            y_pred_val_f,
            normalize='true',
            values_format=".2f",
            ax=ax[1], cmap=plt.cm.Greens
        )

        f1_score_val = f1_score(y_true_val_f, y_pred_val_f, average='weighted')
        recall_score_val = recall_score(y_true_val_f, y_pred_val_f, average='weighted')
        prec_score_val = precision_score(y_true_val_f, y_pred_val_f, average='weighted')

        disp_2.ax_.set_title(f"{model} - Validation Dataset\nF1: {str(f1_score_val.round

    f1_score_pred = f1_score(y_true_train_f, y_pred_train_f, average='weighted')
    recall_score_pred = recall_score(y_true_train_f, y_pred_train_f, average='weighted')
    prec_score_pred = precision_score(y_true_train_f, y_pred_train_f, average='weighted'
    title = None
```

```
    if data_set is not None:
        title = data_set
    else:
        title = 'Training'
    disp_1.ax_.set_title(f"{model} - {title} Dataset\nF1: {str(f1_score_pred.round(3))}\

    fig.tight_layout()
```

In [354…

```
def plot_proportions(true, pred, val):
    fig, ax = plt.subplots(layout='constrained', figsize=(8, 4))

    y_train_labels_counts = np.unique(get_labels(true), return_counts=True)
    y_pred_labels_counts = np.unique(get_labels(pred), return_counts=True)
    y_val_label_counts = None

    if val is not None:
        y_val_label_counts = np.unique(get_labels(val), return_counts=True)

    def get_vals(list_in):
        return_y = [0,0,0,0,0]
        ind = 0
        for i in range(5):
            if len(np.where(list_in[0][ind] == i + 1)[0]) > 0:
                return_y[i] = list_in[1][ind]
                ind+=1
        return return_y

    y_1 = get_vals(y_pred_labels_counts)
    y_2 = None
    if val is not None:
        y_2 = get_vals(y_val_label_counts)

    rating = ("1", "2", "3", '4', '5')
    pop = None
    if val is not None:
        pop = {
            'true proportions': np.round((y_train_labels_counts[1]/sum(y_train_labels_co
            'pred proportions': np.round((y_1/sum(y_1)), 2),
            'val proportions': np.round((y_2/sum(y_2)), 2),
        }
    else:
        pop = {
            'true proportions': np.round((y_train_labels_counts[1]/sum(y_train_labels_co
            'pred proportions': np.round((y_1/sum(y_1)), 2)
        }

    x = y_train_labels_counts[0]  # the label locations
    width = 0.25  # the width of the bars
    multiplier = 0
    color = ['salmon', 'skyblue', 'lightgreen']
    c_ind = 0
    for att, vals in pop.items():
        offset = width * multiplier
        rects = ax.bar(x + offset, vals, width, label=att, color=color[c_ind])
        ax.bar_label(rects, padding=3)
        multiplier += 1
        c_ind += 1

    # Add some text for labels, title and custom x-axis tick labels, etc.
    ax.set_ylabel('Predictions per rating (scaled)')
    ax.set_title('Rating')
    ax.set_xticks(x + width, rating)
    ax.legend(loc='upper left', ncols=1)
    #ax.set_ylim(0, 250)
```

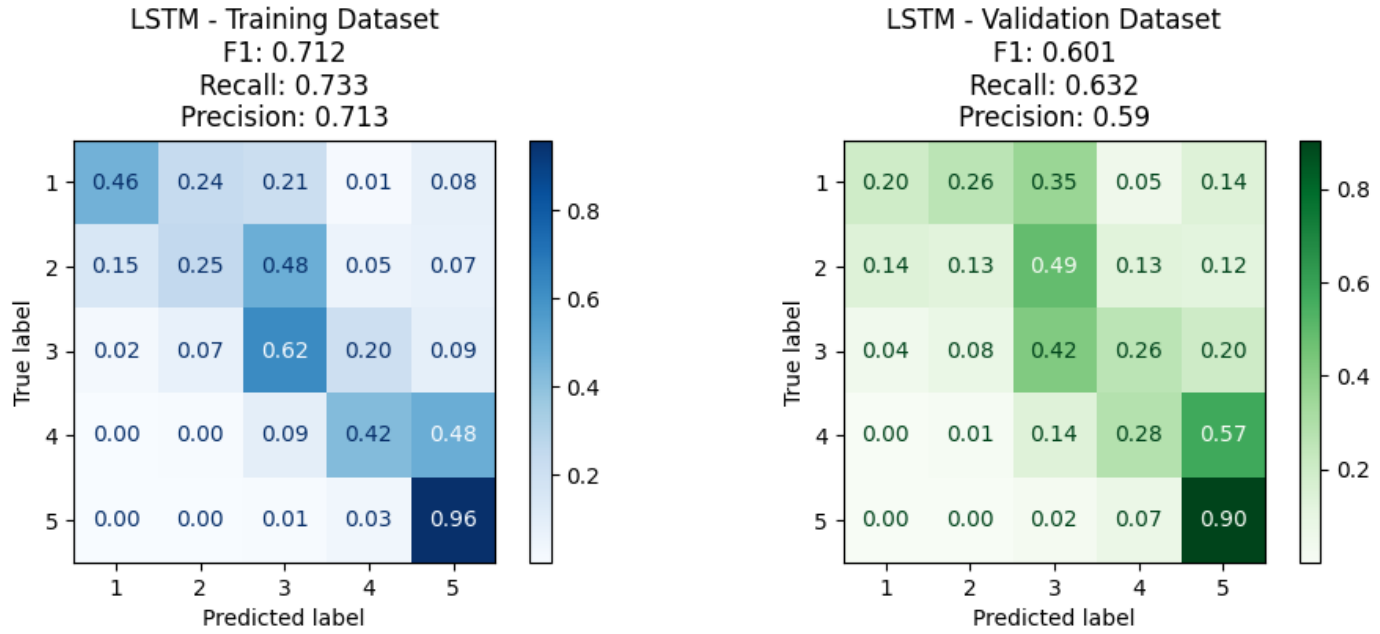
```
    plt.show()
```

Assessment of LSTM model

In [53]:
```
y_pred_train_lstm = final_lstm_model.predict(X_train)
y_pred_val_lstm = final_lstm_model.predict(X_val)
```

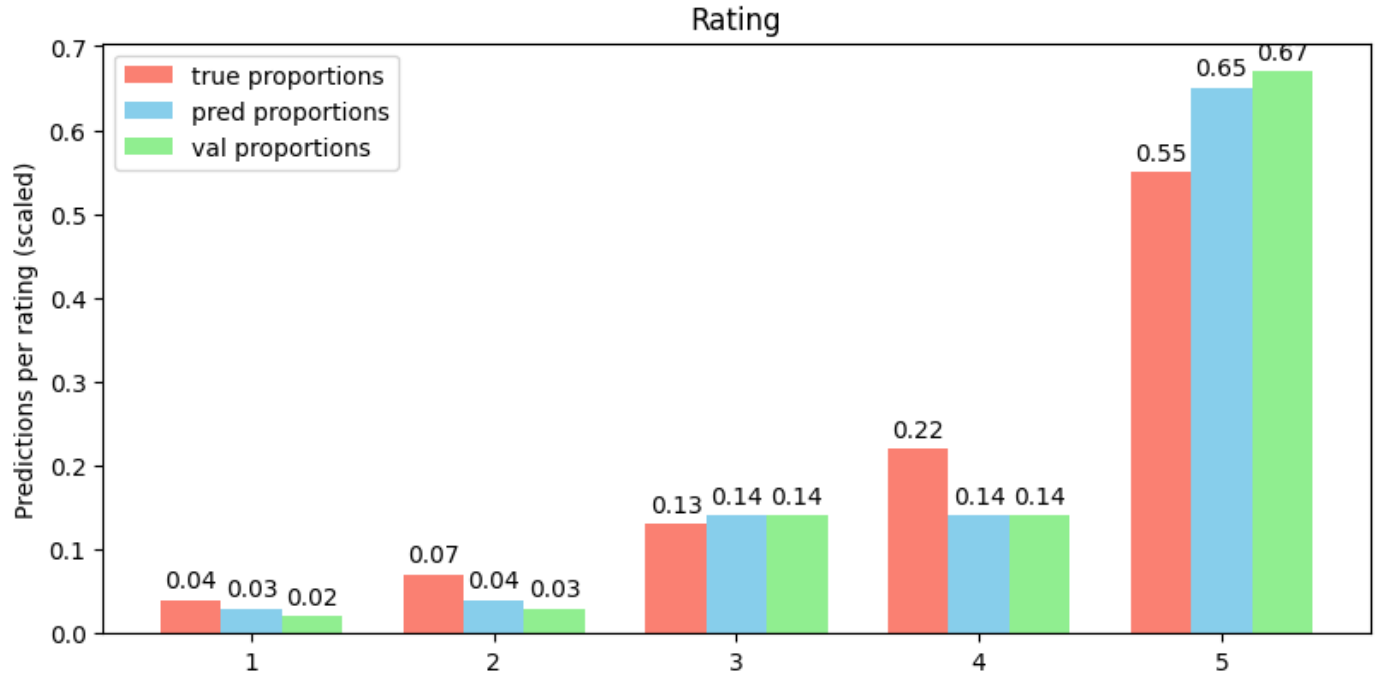
```
397/397 [==============================] - 2s 5ms/step
170/170 [==============================] - 1s 4ms/step
```

In [188…
```
displayConfusionMatrix(y_train, y_pred_train_lstm,  model='LSTM', y_true_val=y_val, y_pr
```



In [355…
```
plot_proportions(y_train, y_pred_train_lstm, y_pred_val_lstm)
```

```
C:\Users\kgrit\AppData\Local\Temp\ipykernel_25636\1159037735.py:15: DeprecationWarning:
Calling nonzero on 0d arrays is deprecated, as it behaves surprisingly. Use `atleast_1d
(cond).nonzero()` if the old behavior was intended. If the context of this warning is of
the form `arr[nonzero(cond)]`, just use `arr[cond]`.
  if len(np.where(list_in[0][ind] == i + 1)[0]) > 0:
```



Here you can see that the best final model did very well accuratly labeling review texts that were rated 5 star and did decently overall. It was able to score highest in the correct label categroies for 3 of the 5 labels, and had a little cross over in neighboring labels. For example, for reviews that were truly 4 star reviews, the model predicted their labels as either 3, 4, or 5, with most being 4 or 5. None or practically none of the 4 star reviews were ever predicted to be 1 or 2 star reviews.

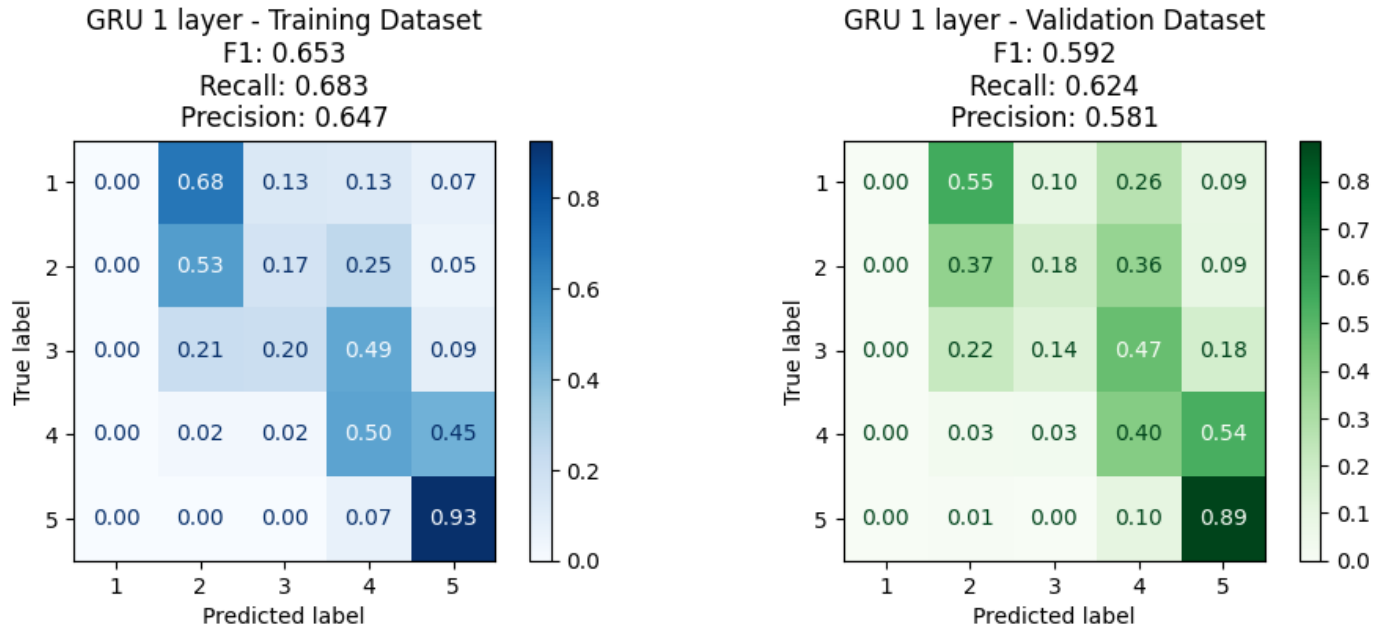
Assessment of 1-layer GRU model

In [199…
```
y_pred_train_gru_1_layer = final_gru_1_layer_model.predict(X_train)
y_pred_val_gru_1_layer = final_gru_1_layer_model.predict(X_val)
```

```
397/397 [==============================] - 1s 3ms/step
170/170 [==============================] - 0s 3ms/step
```

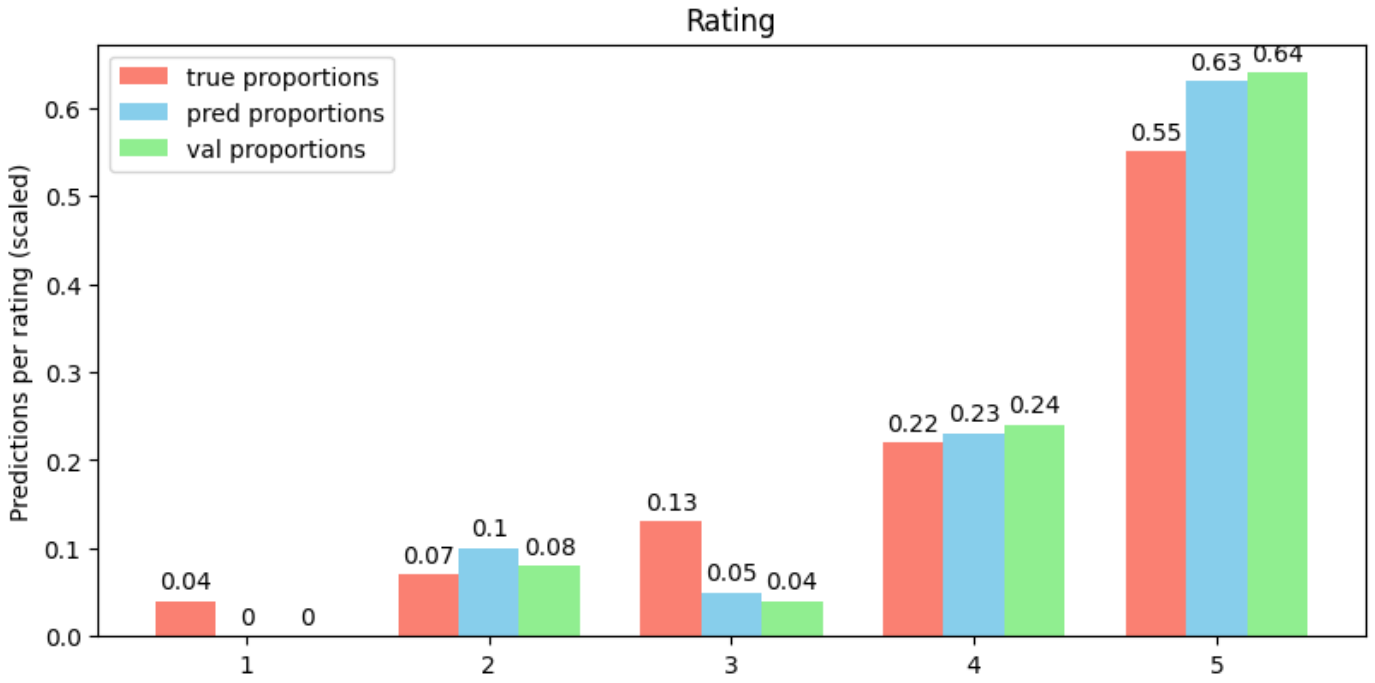
In [201…
```
displayConfusionMatrix(y_train, y_pred_train_gru_1_layer, model='GRU 1 layer', y_true_va
```

```
c:\Users\kgrit\miniconda3\envs\tf\lib\site-packages\sklearn\metrics\_classification.py:1
509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels wit
h no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\kgrit\miniconda3\envs\tf\lib\site-packages\sklearn\metrics\_classification.py:1
509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels wit
h no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



In [356…
```
plot_proportions(y_train, y_pred_train_gru_1_layer, y_pred_val_gru_1_layer)
```

```
C:\Users\kgrit\AppData\Local\Temp\ipykernel_25636\1159037735.py:15: DeprecationWarning:
Calling nonzero on 0d arrays is deprecated, as it behaves surprisingly. Use `atleast_1d
(cond).nonzero()` if the old behavior was intended. If the context of this warning is of
the form `arr[nonzero(cond)]`, just use `arr[cond]`.
  if len(np.where(list_in[0][ind] == i + 1)[0]) > 0:
```



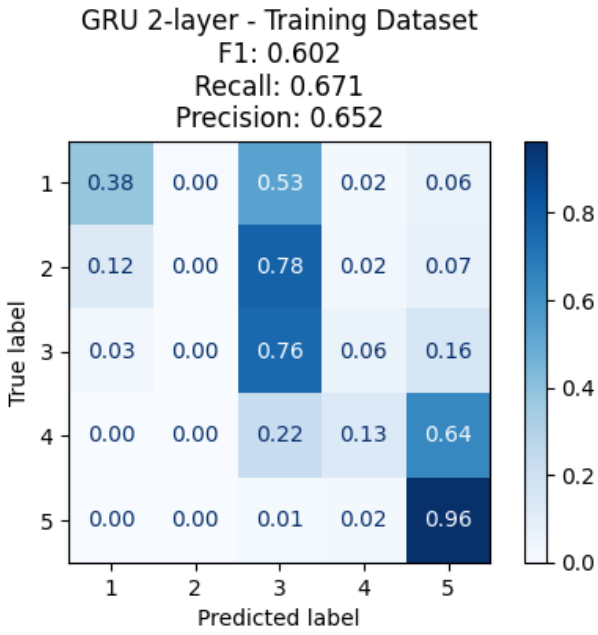
Here you can see the 1-layer GRU model predicted that most of the labels would be either 2, 4, or 5. This model also did well with predicting what was actaully a 5 star review but stuggled with the others. A potential reason why it was able to get as many 2 and 4 star reviews correct as it did was because its fitting and focused on 2, 4, and 5 star labels at the expense of 1 and 3 star reviews.

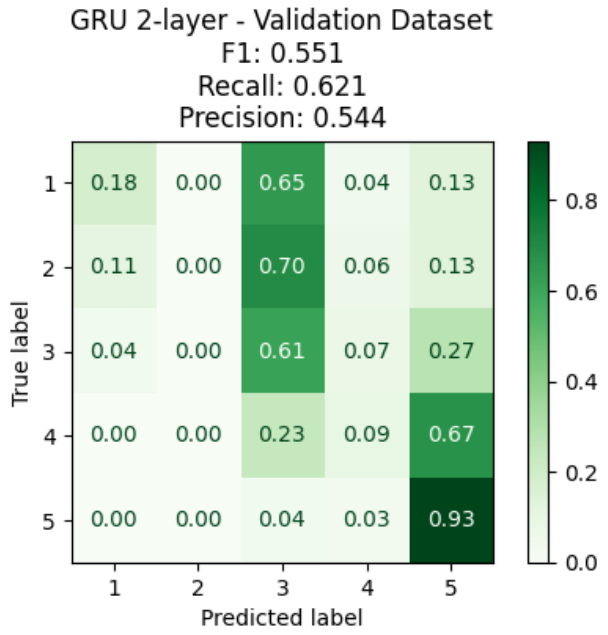
Assessment of 2-layer GRU model

In [161…
```
y_pred_train_gru = final_gru_model.predict(X_train)
y_pred_val_gru = final_gru_model.predict(X_val)
```

```
397/397 [==============================] - 2s 4ms/step
170/170 [==============================] - 1s 3ms/step
```

In [203…
```
displayConfusionMatrix(y_train, y_pred_train_gru, y_true_val=y_val, y_pred_val=y_pred_va
```

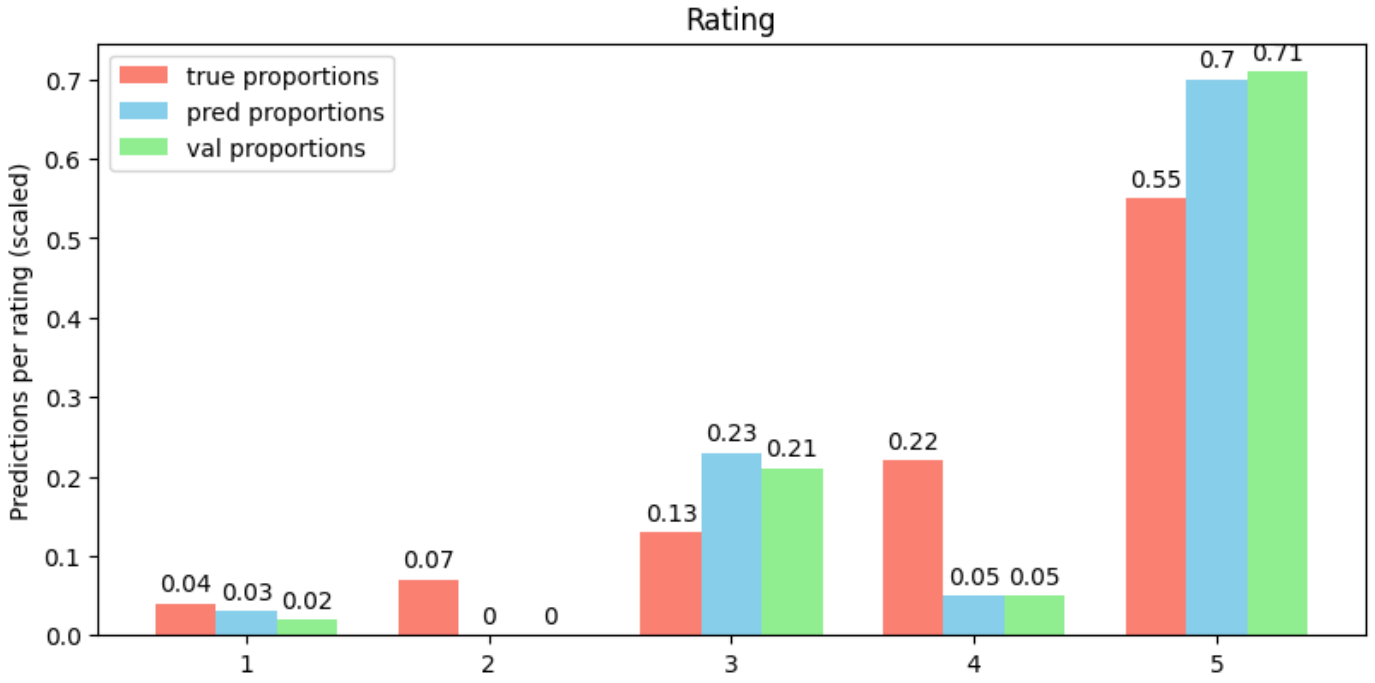




In [357…
```
plot_proportions(y_train, y_pred_train_gru, y_pred_val_gru)
```

```
C:\Users\kgrit\AppData\Local\Temp\ipykernel_25636\1159037735.py:15: DeprecationWarning:
```

```
Calling nonzero on 0d arrays is deprecated, as it behaves surprisingly. Use `atleast_1d
(cond).nonzero()` if the old behavior was intended. If the context of this warning is of
the form `arr[nonzero(cond)]`, just use `arr[cond]`.
  if len(np.where(list_in[0][ind] == i + 1)[0]) > 0:
```



As with the 1-layer GRU model, the 2-layer GRU focused on getting just 3 layers right. It predicted that most of the labels would be either 1, 3, or 5. As before, the model also did well with predicting what was actaully a 5 star review but stuggled with the others.

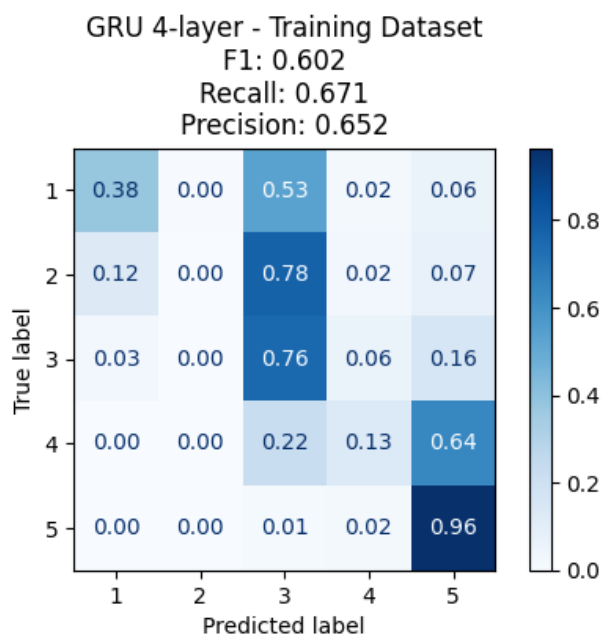
Assessment of GRU 4-layer model

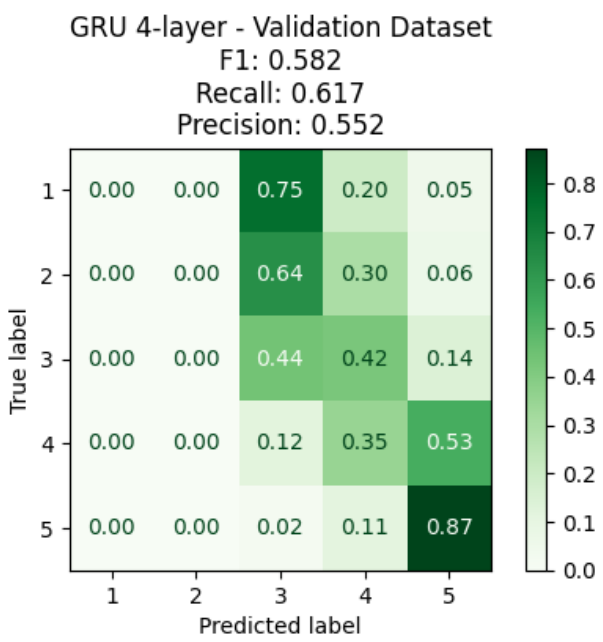
In [67]:
```
y_pred_train_gru_4L = final_gru_4_L_model.predict(X_train)
y_pred_val_gru_4L = final_gru_4_L_model.predict(X_val)
```

```
397/397 [==============================] - 11s 26ms/step
170/170 [==============================] - 4s 25ms/step
```

In [204…
```
displayConfusionMatrix(y_train, y_pred_train_gru, y_true_val=y_val, y_pred_val=y_pred_va
```

```
c:\Users\kgrit\miniconda3\envs\tf\lib\site-packages\sklearn\metrics\_classification.py:1
509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels wit
h no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

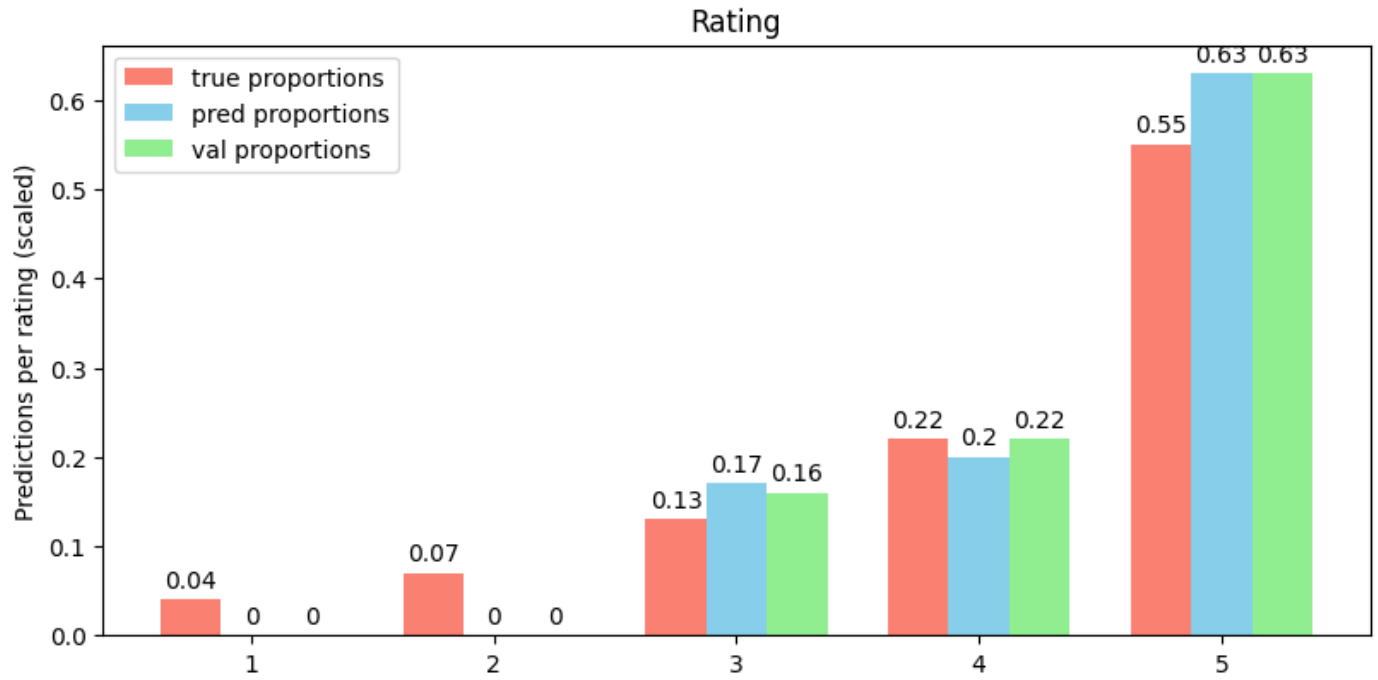




In [358…

```
plot_proportions(y_train, y_pred_train_gru_4L, y_pred_val_gru_4L)
```

```
C:\Users\kgrit\AppData\Local\Temp\ipykernel_25636\1159037735.py:15: DeprecationWarning:
Calling nonzero on 0d arrays is deprecated, as it behaves surprisingly. Use `atleast_1d
(cond).nonzero()` if the old behavior was intended. If the context of this warning is of
the form `arr[nonzero(cond)]`, just use `arr[cond]`.
  if len(np.where(list_in[0][ind] == i + 1)[0]) > 0:
```



Finally, we have the 4-layer GRU model. As with the 1-layer and 2-layer GRU model, the 4-layer GRU focused on getting just 3 layers right. It predicted that most of the labels would be either 3, 4, or 5. As before, the model also did well with predicting what was actaully a 5 star review but stuggled with the others.

Final test prediction and score for best performing model

The test data set will now be fed into the pretrained tokenizer, formated for the Bi-directional LSTM model (it had the best recall score), verified it can be reconstructed into the original cleaned tweets, and passed into the model for predictiong the correct test tweet label.

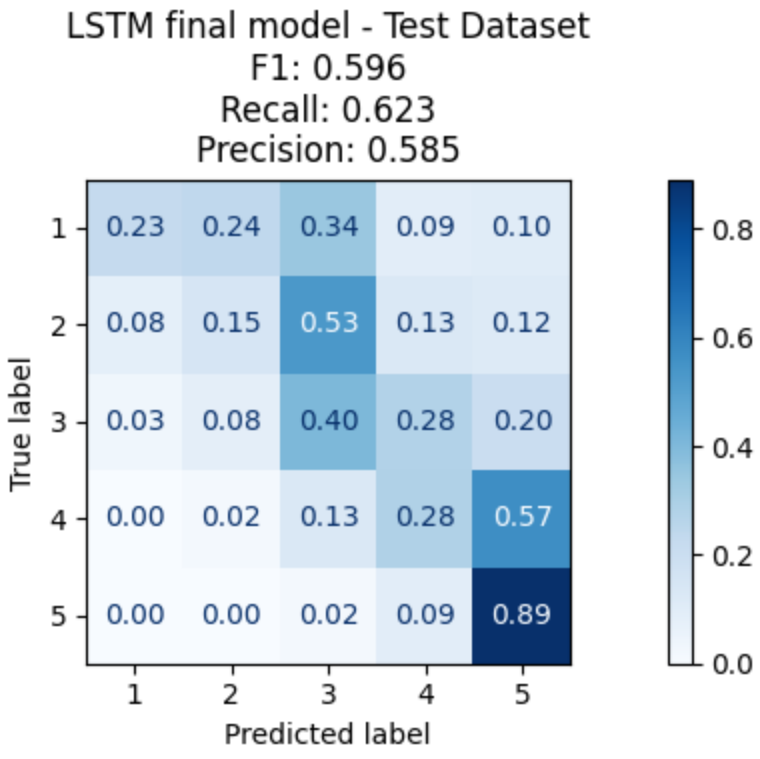
In [166…
```
for i in range(5,8):
    print(f'row {i}')
    print('test:\n    ', tokenizer.sequences_to_texts([final_test[i]]))
    print('original cleaned tweet:\n    ', df_test.iloc[i]['text_clean'])
```

```
row 5
test:
     ['bought green thought bit different super flattering fabric soft comfortable xs ye
t decided keeping work im 54 116 going wear upcoming trip think get lot use rest summer
early fall saw black tempted buy well mentioned love dress']
original cleaned tweet:
     bought green thought bit different super flattering fabric soft comfortable xs yet
decided keeping work im 54 116 going wear upcoming trip think get lot use rest summer ea
rly fall saw black tempted buy well mentioned love dress
row 6
test:
     ['bought dress use maternity photos perfect love flow skirt elastic waistband worke
d perfect able fit size 2 normally size 0 hope able wear pregnancy well worried bust are
a small tight fit currently still worked']
original cleaned tweet:
     bought dress use maternity photos perfect love flow skirt elastic waistband worked
perfect able fit size 2 normally size 0 hope able wear pregnancy well worried bust area
small tight fit currently still worked
row 7
test:
     ['piece fits true size flattering on sometimes worry stomach perfectly flat issue a
ll probably intricate lace details fabric substantial enough feel like itll get hole mul
tiple wears']
original cleaned tweet:
     piece fits true size flattering on sometimes worry stomach perfectly flat issue all
probably intricate lace details fabric substantial enough feel like itll get hole multip
le wears
```

In [170…
```
y_pred_test = final_lstm_model.predict(X_test)
```

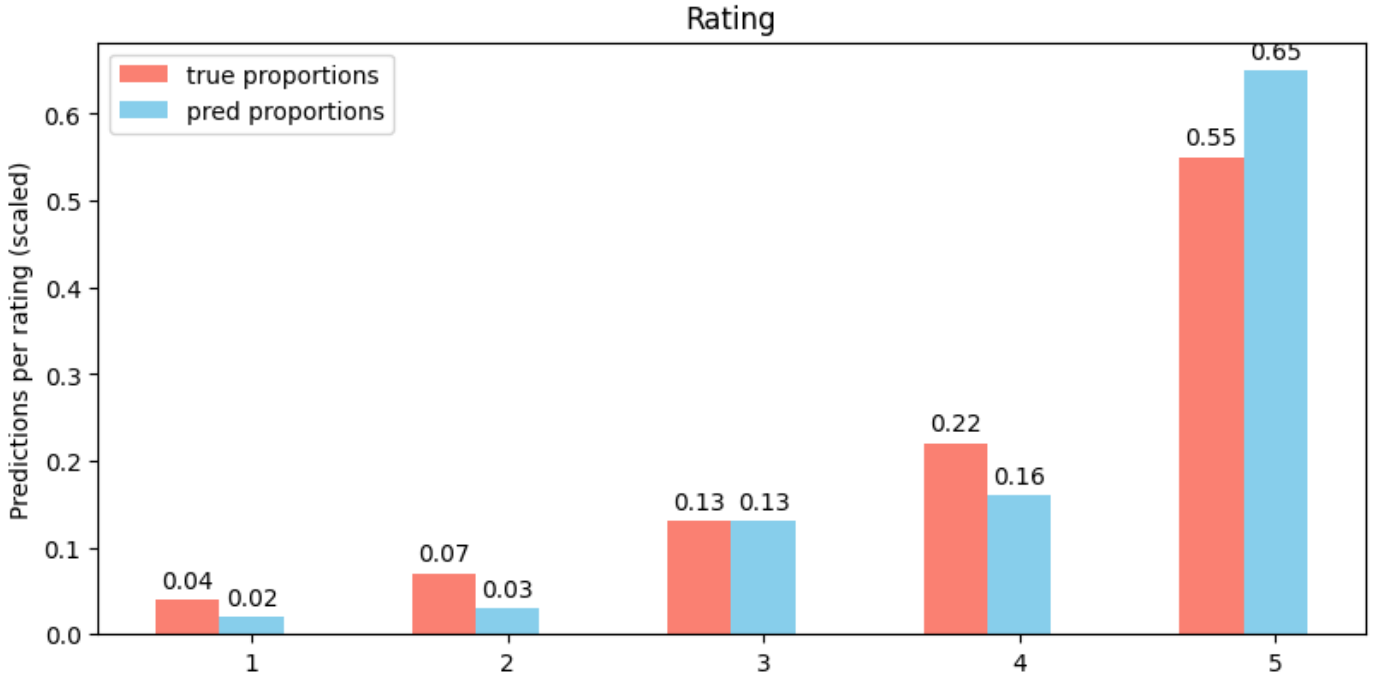
```
  1/142 [..............................] - ETA: 15s142/142 [==============================
==] - 1s 5ms/step
```

In [189…
```
displayConfusionMatrix(y_test, y_pred_test, model='LSTM final model', data_set='Test')
```



In [359…
```
plot_proportions(y_train, y_pred_test, None)
```

```
C:\Users\kgrit\AppData\Local\Temp\ipykernel_25636\1159037735.py:15: DeprecationWarning:
Calling nonzero on 0d arrays is deprecated, as it behaves surprisingly. Use `atleast_1d
(cond).nonzero()` if the old behavior was intended. If the context of this warning is of
the form `arr[nonzero(cond)]`, just use `arr[cond]`.
  if len(np.where(list_in[0][ind] == i + 1)[0]) > 0:
```



Here we can see similar behavior of the model when it was predicting labels for the training and validation data set reviews. Importanly, it is not over-prioritizing geting a smaller set of the labels accurate, as the GRU models were tending to do. The model was sstill nnot able to make decent predictions on the test data set however. As expected, it did well with accuratley labeling the 5 star reviews but struggled with the others. The only case where it scored highest on the actual label was with 3 star reviews

Conclusion

General takeways

From the three model architectures test, the 2 layer LSTM model did the best, though not by much. The F1 score of its final prediction was only 0.596, ands you can see in the confusion matrix above, the model was not able to reach above 50% label accuracy for 4 of the 5 labels present. It did quite well however determining what was truely a 5 star review and what was not, corerctly labeling 89% of them. This is likely because of the number of 5 star reviews present in the data set was both large and disproportionally larger than the counts of the other review ratings in the data. The histrograms displayed in the EDA section revealed that around 50% of the reviews in the data were 5 star reviews.

One big takeway from this problem attempt is that just adding more layers of a given unit (GRU, LSTM, etc) or more layers at all to an RNN is not going necessarily going to improve any of the metrics used to assess model predictive power or accuracy. For instance, one of the problematic features of RNN's layers is vanishing/exploding gradient problem, which can be addressed by adding dropout layers with different dropout rates between cells. In the architectures tested here, 3 of the 4 optimal sets of hyperparameters had some dropout layers with a 0.0 dropout rate, indicating they allowed the previous and following layers to

remain fully connecetd. Additionally, when testing the affect of additional layers of GRU units, adding more layers did not improve model performance, even with parameter tuning

Model performace helpers and detractors

On previous iterations of this problem challenge, I set the models to use accuracy score in the model compiler. I found this did not help because even though the accuracy scores and F1 scores did improve, the models did poorly when it came to predicting labels for reviews that were not 5 stars. By changing the scoring metric to multi-label AUC, I was able to modestly improve the model performances.

A significant helper in improving model performance was the use of hyperparameter tuning. Each model architecture tested here was assessed first using 50 randomly selected sets of hyperparameters, allowing for a significant

Another thing that had a positive impact on model performance was the Word2Vec word embedding process. After testing out a few others such as TF-IDF and Bag-of-words, I found that Word2Vec operations produced a much smaller input for the models and allowed them to be trained very fast, orders of magnitude faster than with the other word embedding stategizes as I had implimented them.

Possible future improvements

In the future, performing exhaustive hyperparameter space tuning would likely improve model performance, or at least confirm that optimal sets of parameters were found in this notebook. In this case, 50 sets of parameters were randomly selected from a larger space of parameters. This was not close to exhaustive, as for the 4-layer GRU model, 50 sets out of a total of 1,600, or about 3%.

Another possible way to improve the model performances would be to run more validation trials in the tuning process. Keras tuners allow you to train a batch of models for a given set of hyperparameters, then take the best scoring model and compare that to the other best performing hyperparameter sets. Runing 2 or more trials instead of just one would provide a more robust assessment of a given set of hyperparameters preformance with a model architecture.

Acknowledgements

1. Koo Ping Shung (2018) Accuracy, Precision, Recall or F1?, Towards Data Science. Available at: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.

2. Akash Deep (2020) Understanding NLP Keras Tokenizer Class Arguments with example. Medium. Available at: https://medium.com/analytics-vidhya/understanding-nlp-keras-tokenizer-class-arguments-with-example-551c100f0cbd.

In [ ]:

In [ ]: