



Search

You are viewing the last successful run of this notebook. Click [here](#) to see the current version with an error.

Dismiss



AADARSH KUMAR GUPTA · 2 MO AGO · 183 VIEWS



0



Milestone Neural Network

Python · Twitter Sentiment Analysis

Notebook Input Output Logs Comments (0) Settings

Run

3901.7s

Version 7 of 9

Add Tags

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

/kaggle/input/twitter-entity-sentiment-analysis/twitter_validation.csv
/kaggle/input/twitter-entity-sentiment-analysis/twitter_training.csv
```

In [2]:

```
# Import pandas
import pandas as pd

# Import training data
df_train = pd.read_csv('/kaggle/input/twitter-entity-sentiment-analysis/twitter_training.csv')

# Print few random entries
df_train.sample(5)
```

Out[2]:

	2401	Borderlands	Positive	im getting on borderlands and i will murder you all ,
31698	7442	LeagueOfLegends	Irrelevant	Thanks for mentioning DragonesAecho.
17712	9837	PlayStation5(PSS)	Positive	Looks really nice
15063	2979	Dota2	Irrelevant	and Goodbye Rog
35631	8118	Microsoft	Positive	2 S1 mini consoles Pico 80 projectors @CanonUS...
19867	12601	WorldOfCraft	Neutral	Another commission for my beautiful @RendDuo;...

In [3]:

```
df_train.rename(columns={'2401': 'id', 'Borderlands': 'game','Positive':'sentiment','im getting on borderlands and i will murder you all ','text'}, inplace=True)
```

In [4]:

```
df_train.sample(5)
```

Out[4]:

	id	game	sentiment	text
31247	7367	LeagueOfLegends	Negative	Attention: due to fear of getting coronavirus ...
18509	9973	PlayStation5(PSS)	Negative	You duhh.
70618	10897	TomClancysGhostRecon	Negative	@UbisoftSupport The @AskPlayStation internet i...
51636	10464	RedDeadRedemption(RDR)	Neutral	Red dead Redemption 2! youtb.be / Wamhb5tUKGw

EDA

```
In [5]: # Checking the number of rows and columns
print("The DataFrame has " + str(df_train.shape[0]) + " samples and " + str(df_train.shape[1]) + " columns")
```

The DataFrame has 74681 samples and 4 columns

```
In [6]: # Check duplicates
print("Duplicate entries in the dataset: " + str(df_train.duplicated().sum()))
```

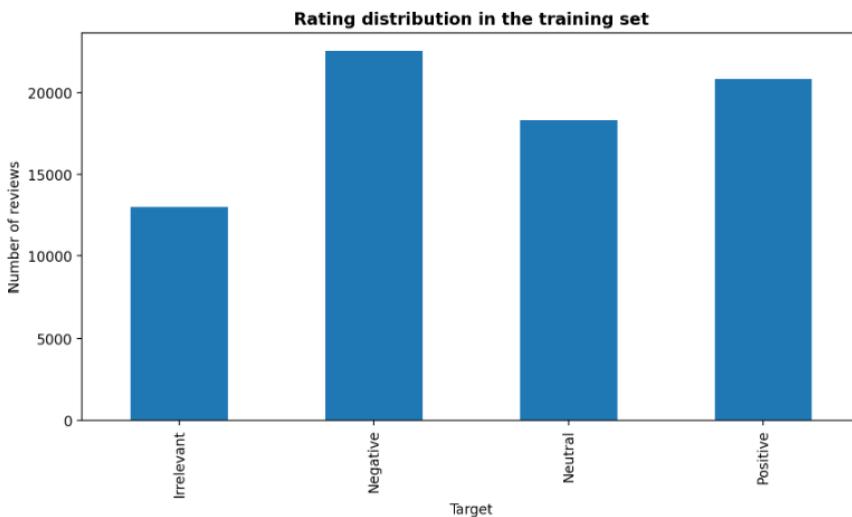
Duplicate entries in the dataset: 2700

```
In [7]: # Check null values and data type of each column
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74681 entries, 0 to 74680
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype  
---  --  
0   id        74681 non-null   int64  
1   game      74681 non-null   object  
2   sentiment 74681 non-null   object  
3   text       73995 non-null   object  
dtypes: int64(1), object(3)
memory usage: 2.3+ MB
```

```
In [8]: import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# Check target balance
df_train['sentiment'].value_counts().sort_index().plot.bar(figsize=(10,5))
plt.title('Rating distribution in the training set', fontweight="bold")
plt.xlabel('Target')
plt.ylabel('Number of reviews')
plt.show()
```



```
In [9]: # Drop few columns
df_train = df_train.drop(['id'],axis=1)
```

```
In [10]: import re
from nltk.corpus import stopwords

# Create a stopwords set adding some personal 'words'
```

```

stopwords_english = set(stopwords.words('english'))
my_stopwords = set(["http", "s", "n't", "'m", "'re", "'ve"])

stopwords_english.update(my_stopwords)

def preprocess_review(text):
    # Convert to lower case
    text = text.lower()

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Remove punctuation
    text = re.sub(r'[^w\s]', '', text)

    # Split text into tokens
    tokens = text.split()

    # Filter tokens
    clean_tokens = [tok for tok in tokens if tok not in stopwords_english and len(tok) > 1]

    # Join tokens into a string
    clean_text = ' '.join(clean_tokens)

    return clean_text

```

```

/opt/conda/lib/python3.10/site-packages/scipy/_init_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5)
warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"

```

There are multiple ways to clean text, including stemming and lemmatization. Although these techniques can enhance the outcome, they also slow down the function considerably, especially for larger datasets like this one. For instance, my previous function using these techniques took more than an hour to train the dataset, whereas the new solution takes a little over a minute. Therefore, I opted for a faster function even though it may result in slightly less accuracy.

Let's check it out:

```

In [11]: # Get one review as sample
sample = df_train['text'][5]

print('ORIGINAL REVIEW:  ' + sample + '\n')
print('-----\n')
print('WITH PROCESSING:  ' + preprocess_review(sample))

```

```

ORIGINAL REVIEW:  So I spent a few hours making something for fun. . . If you don't know I am a HUGE @Borderlands fan and
Maya is one of my favorite characters. So I decided to make myself a wallpaper for my PC. . Here is the original image ver
sus the creation I made :) Enjoy! pic.twitter.com/mLsI5wf9Jg
-----
```

```

WITH PROCESSING:  spent hours making something fun dont know huge borderlands fan maya one favorite characters decided m
ake wallpaper pc original image versus creation made enjoy pictwittercommliwfjg
-----
```

```

In [12]: df_train.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74681 entries, 0 to 74680
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype  
---  -- 
 0   game        74681 non-null   object 
 1   sentiment   74681 non-null   object 
 2   text         73995 non-null   object 
dtypes: object(3)
memory usage: 1.7+ MB

```

```

In [13]: df_train.dropna(subset=['text'], inplace=True)

```

```

In [14]: %time

# Preprocess training data adding new column
df_train["clean_text"] = df_train["text"].apply(preprocess_review)

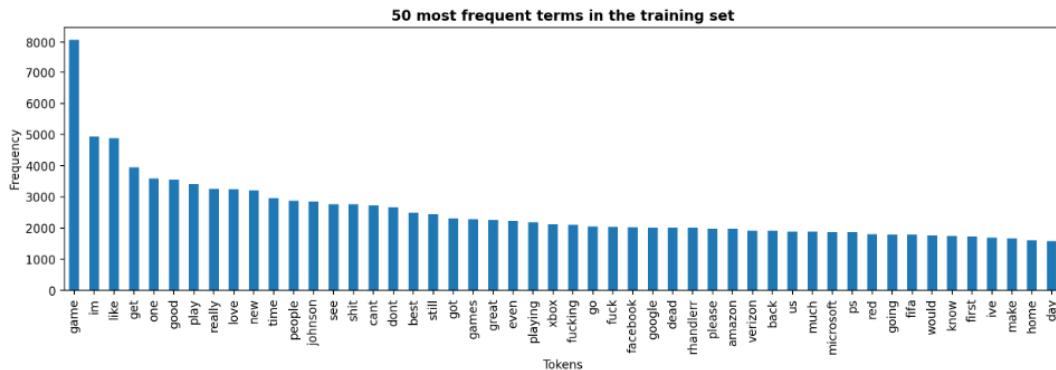
```

```

CPU times: user 1.6 s, sys: 11 ms, total: 1.61 s
Wall time: 1.61 s

```

```
In [15]: # Get 50 most frequent words in the training set
frequent_words = pd.Series(' '.join(df_train.clean_text).split()).value_counts()[:50]
frequent_words.plot.bar(figsize=(15,4))
plt.title('50 most frequent terms in the training set', fontweight="bold")
plt.xlabel('Tokens')
plt.ylabel('Frequency')
plt.show()
```



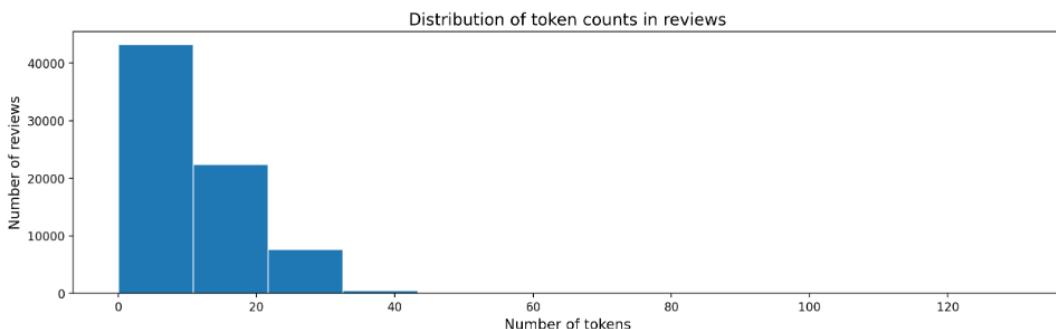
```
In [16]: # I found that the variable 'frequent_words' takes up about 6 Gb of RAM! Better clean it up!
import gc

del frequent_words

gc.collect()
```

```
Out[16]: 10706
```

```
In [17]: # Plot the number of tokens in cleaned reviews
plt.figure(figsize=(15,4))
plt.hist(df_train['clean_text'].apply(lambda x:len(x.split())), bins=12, edgecolor='white')
plt.xlabel('Number of tokens', fontsize=12)
plt.ylabel('Number of reviews', fontsize=12)
plt.title('Distribution of token counts in reviews', fontsize=14)
plt.show()
```



```
In [18]: # Filter rows with less than 250 tokens
less_than_250 = df_train['clean_text'].apply(lambda x: len(x.split())) < 250 # Check by getting True and False values

# Print percentage
print(f"({sum(less_than_250) / len(df_train) * 100}%.2f) % of rows have less than 250 tokens.")
```

```
100.00% of rows have less than 250 tokens.
```

```
In [19]: # Remove original text column
df_train = df_train.drop(columns=['text'])

# Get few entries to check operation
df_train.head()
```

```
Out[19]:
```

	game	sentiment	clean_text
0	Borderlands	Positive	coming borders kill
1	Borderlands	Positive	im getting borderlands kill

2	Borderlands	Positive	im coming borderlands murder
3	Borderlands	Positive	im getting borderlands murder
4	Borderlands	Positive	im getting borderlands murder

Test Set

Importing and processing the text data as did in the training set

```
In [20]: # Import training data
df_test = pd.read_csv('/kaggle/input/twitter-entity-sentiment-analysis/twitter_validation.csv')

# Print a random row
df_test.sample()
```

Out[20]:

	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling for motivation to go for a run the other day, which has been translated by Tom's great auntie as 'Hayley can't get out of bed' and told to his grandma, who now thinks I'm a lazy, terrible person 🤦
274	6851	johson&johnson	Neutral	'Skunkworks' at Johnson & Johnson is rushing t...

```
In [21]: df_test.rename(columns={'3364': 'id', 'Facebook': 'game', 'Irrelevant':'sentiment', 'I mentioned on Facebook that I was struggling for motivation to go for a run the other day, which has been translated by Tom's great auntie as 'Hayley can't get out of bed' and told to his grandma, who now thinks I'm a lazy, terrible person 🤦':'text'}, inplace=True)
```

In [22]:

```
%time

# Remove uncecessary columns
df_test = df_test.drop(['id'],axis=1)

# Preprocess training data adding new column
df_test["clean_text"] = df_test["text"].apply(preprocess_review)

# Remove original text column
df_test = df_test.drop(columns=['text'])

# Get few entries to check cleaning operation
df_test.head()
```

CPU times: user 30.3 ms, sys: 54 µs, total: 30.4 ms
Wall time: 30.1 ms

Out[22]:

	game	sentiment	clean_text
0	Amazon	Neutral	bbc news amazon boss jeff bezos rejects claims...
1	Microsoft	Negative	microsoft pay word functions poorly samsungus ...
2	CS-GO	Negative	csgo matchmaking full closet hacking truly awf...
3	Google	Neutral	president slapping americans face really commi...
4	FIFA	Negative	hi eahelp ive madeleine mccann cellar past yea...

In [23]:

```
#Label_encoding For sentiments.
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df_train['sentiment']=label_encoder.fit_transform(df_train['sentiment'])
df_test['sentiment']=label_encoder.fit_transform(df_test['sentiment'])
```

Split the data

In [24]:

```
# Create X, y arrays
from sklearn.model_selection import train_test_split

# Training data
X_train = df_train["clean_text"].values
y_train = df_train["sentiment"].values

# Split into train/test sets
X_tr, X_va, y_tr, y_va = train_test_split(X_train, y_train, test_size=0.2, random_state=0)

# Test data
X_te = df_test["clean_text"].values
```

```

print("Training data:", X_tr.shape, y_tr.shape)
print("Validation data:", X_va.shape, y_va.shape)
print("Test data:", X_te.shape)

```

```

Training data: (59196,) (59196,)
Validation data: (14799,) (14799,)
Test data: (999,)

```

Model Part

Given the dataset's size and my preference for using a Neural Network model, it's certainly worthwhile to employ an accelerator. Below are the default setup settings for enabling the TPU (Tensor Processing Unit).

```

In [25]:
import warnings
warnings.filterwarnings("ignore")

import tensorflow as tf

try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)

    strategy = tf.distribute.experimental.TPUStrategy()
except ValueError:
    strategy = tf.distribute.get_strategy()
    print('Number of replicas:', strategy.num_replicas_in_sync)

```

```
Number of replicas: 1
```

```

In [26]:
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver() # TPU detection
except ValueError:
    tpu = None
    gpus = tf.config.experimental.list_logical_devices("GPU")

if tpu:
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
    print('Running on TPU ', tpu.cluster_spec().as_dict()['worker'])
elif len(gpus) > 1:
    strategy = tf.distribute.MirroredStrategy([gpu.name for gpu in gpus])
    print('Running on multiple GPUs ', [gpu.name for gpu in gpus])
elif len(gpus) == 1:
    strategy = tf.distribute.get_strategy()
    print('Running on single GPU ', gpus[0].name)
else:
    strategy = tf.distribute.get_strategy()
    print('Running on CPU')
print("Number of accelerators: ", strategy.num_replicas_in_sync)

```

```
Running on CPU
Number of accelerators: 1
```

Now, this code uses the Keras module from TensorFlow to preprocess text data. Specifically, it uses the 'Tokenizer' function to convert the text into a sequence of integers, and the 'pad_sequences' function to ensure that each sequence has the same length.

```

In [27]:
%%time

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Tokenize and pad the sequences
tokenizer = Tokenizer(num_words=20000)
tokenizer.fit_on_texts(X_tr)

max_seq_length = 250

X_tr_seq = tokenizer.texts_to_sequences(X_tr)
X_tr_seq = pad_sequences(X_tr_seq, maxlen=max_seq_length)

X_va_seq = tokenizer.texts_to_sequences(X_va)
X_va_seq = pad_sequences(X_va_seq, maxlen=max_seq_length)

X_te_seq = tokenizer.texts_to_sequences(X_te)
X_te_seq = pad_sequences(X_te_seq, maxlen=max_seq_length)

```

```
CPU times: user 3.07 s, sys: 45.1 ms, total: 3.11 s
Wall time: 3.12 s
```

```
In [28]: X_tr_seq
```

```
Out[28]: array([[ 0,  0,  0, ..., 557, 141, 19845],
 [ 0,  0,  0, ...,  0,  0, 236],
 [ 0,  0,  0, ..., 3696, 252,  34],
 ...,
 [ 0,  0,  0, ...,  80,  49, 204],
 [ 0,  0,  0, ..., 721, 721, 10729],
 [ 0,  0,  0, ...,  23, 3238, 4410]], dtype=int32)
```

```
In [29]: X_va_seq
```

```
Out[29]: array([[ 0,  0,  0, ..., 54, 3294, 864],
 [ 0,  0,  0, ..., 140, 17, 4517],
 [ 0,  0,  0, ..., 80, 469, 5948],
 ...,
 [ 0,  0,  0, ..., 394, 125, 1296],
 [ 0,  0,  0, ..., 1442, 658, 2281],
 [ 0,  0,  0, ...,  0,  0, 238]], dtype=int32)
```

```
In [30]: X_te_seq
```

```
Out[30]: array([[ 0,  0,  0, ..., 1821, 8296, 15319],
 [ 0,  0,  0, ..., 2364, 13835, 6435],
 [ 0,  0,  0, ..., 568, 1131,   1],
 ...,
 [ 0,  0,  0, ..., 152,  51, 382],
 [ 0,  0,  0, ..., 58, 577, 675],
 [ 0,  0,  0, ..., 36, 713, 4820]], dtype=int32)
```

Let's build the model. There are various approaches available, but I decided to experiment with a combination of several different layers, including a convolutional layer and a bidirectional LSTM layers. Probably these ones may be responsible for slowing down the training time more than the other layers. Anyway, let's evaluate it afterwards:

```
In [31]:
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, LSTM, Bidirectional, Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam

with strategy.scope():
    model = Sequential()
    model.add(Embedding(input_dim=20000, output_dim=128, input_length=250))

    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))

    model.add(Bidirectional(LSTM(units=64, return_sequences=True)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Bidirectional(LSTM(units=128)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(units=32, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(units=6, activation='softmax'))

model.compile(optimizer=Adam(learning_rate = 1e-4),
              loss = 'sparse_categorical_crossentropy',
              metrics=['acc'])

model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
--------------	--------------	---------

```

-----  

embedding (Embedding)      (None, 250, 128)      2560000  

conv1d (Conv1D)            (None, 248, 64)        24640  

max_pooling1d (MaxPooling1D (None, 124, 64)      0  

)  

bidirectional (Bidirectiona (None, 124, 128)      66048  

l)  

batch_normalization (BatchN (None, 124, 128)      512  

ormalization)  

dropout (Dropout)          (None, 124, 128)      0  

bidirectional_1 (Bidirectio (None, 256)          263168  

nal)  

batch_normalization_1 (Batac (None, 256)          1024  

hNormalization)  

dropout_1 (Dropout)         (None, 256)          0  

dense (Dense)              (None, 32)            8224  

batch_normalization_2 (Batac (None, 32)            128  

hNormalization)  

dropout_2 (Dropout)         (None, 32)            0  

dense_1 (Dense)             (None, 6)             198  

=====  

Total params: 2,923,942  

Trainable params: 2,923,110  

Non-trainable params: 832
-----
```

```
In [32]:  
from tensorflow.keras.callbacks import EarlyStopping  
  
# Creates 'EarlyStopping' callback  
earlystopping_cb = EarlyStopping(patience=4, restore_best_weights=True)
```

```
In [33]:  
X_tr_seq.shape  
  
Out[33]:  
(59196, 250)
```

```
In [34]:  
y_tr.shape  
  
Out[34]:  
(59196,)
```

```
In [35]:  
X_va_seq.shape  
  
Out[35]:  
(14799, 250)
```

```
In [36]:  
%time  
  
history = model.fit(X_tr_seq,  
                     y_tr,  
                     validation_data=(X_va_seq, y_va),  
                     callbacks=[earlystopping_cb],  
                     batch_size=70,  
                     epochs=50,  
                     verbose=1,  
                     )  
  
Epoch 1/50  
846/846 [=====] - 451s 522ms/step - loss: 1.7274 - acc: 0.3791 - val_loss: 1.2698 - val_acc: 0.55  
83  
Epoch 2/50  
846/846 [=====] - 437s 517ms/step - loss: 0.8826 - acc: 0.6996 - val_loss: 0.8301 - val_acc: 0.71  
46  
Epoch 3/50  
846/846 [=====] - 452s 534ms/step - loss: 0.4795 - acc: 0.8396 - val_loss: 1.2223 - val_acc: 0.61
```

```

06
Epoch 4/50
846/846 [=====] - 456s 539ms/step - loss: 0.3017 - acc: 0.8983 - val_loss: 0.6231 - val_acc: 0.80
84
Epoch 5/50
846/846 [=====] - 460s 543ms/step - loss: 0.2181 - acc: 0.9265 - val_loss: 0.7239 - val_acc: 0.76
89
Epoch 6/50
846/846 [=====] - 454s 536ms/step - loss: 0.1743 - acc: 0.9404 - val_loss: 0.8835 - val_acc: 0.75
90
Epoch 7/50
846/846 [=====] - 452s 534ms/step - loss: 0.1487 - acc: 0.9487 - val_loss: 0.9836 - val_acc: 0.73
38
Epoch 8/50
846/846 [=====] - 486s 574ms/step - loss: 0.1367 - acc: 0.9516 - val_loss: 2.1465 - val_acc: 0.57
85
CPU times: user 3h 18min 22s, sys: 10min 16s, total: 3h 28min 38s
Wall time: 1h 47s

```

```

In [37]:
import numpy as np

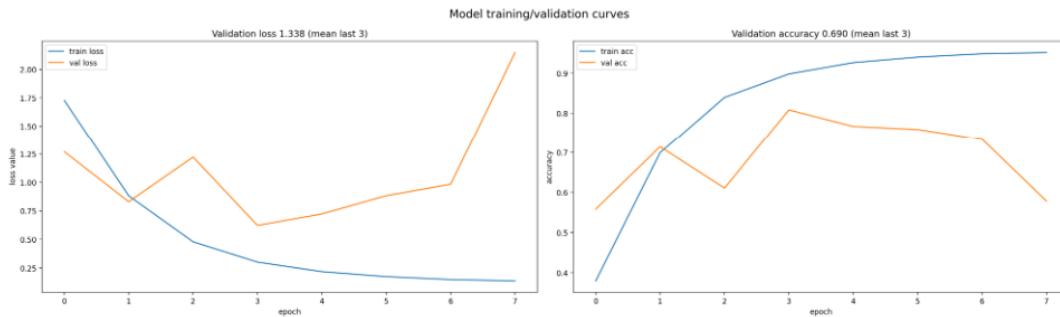
# Create two plots: one for the loss value, one for the accuracy
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))

plt.suptitle('Model training/validation curves', size=15)

# Plot loss values
ax1.plot(history.history["loss"], label="train loss")
ax1.plot(history.history["val_loss"], label="val loss")
ax1.set_title(
    "Validation loss {:.3f} (mean last 3)".format(
        np.mean(history.history["val_loss"][-3:])) # last three values
)
ax1.set_xlabel("epoch")
ax1.set_ylabel("loss value")
ax1.legend()

# Plot accuracy values
ax2.plot(history.history["acc"], label="train acc")
ax2.plot(history.history["val_acc"], label="val acc")
ax2.set_title(
    "Validation accuracy {:.3f} (mean last 3)".format(
        np.mean(history.history["val_acc"][-3:])) # last three values
)
ax2.set_xlabel("epoch")
ax2.set_ylabel("accuracy")
ax2.legend()
plt.tight_layout()
plt.show()

```



```

In [38]:
test_loss, test_acc = model.evaluate(X_va_seq,y_va)

print('Validation loss:', test_loss)
print('Validation accuracy:', test_acc)

```

```

463/463 [=====] - 50s 108ms/step - loss: 0.6231 - acc: 0.8084
Validation loss: 0.6230593919754028
Validation accuracy: 0.8084329962730408

```

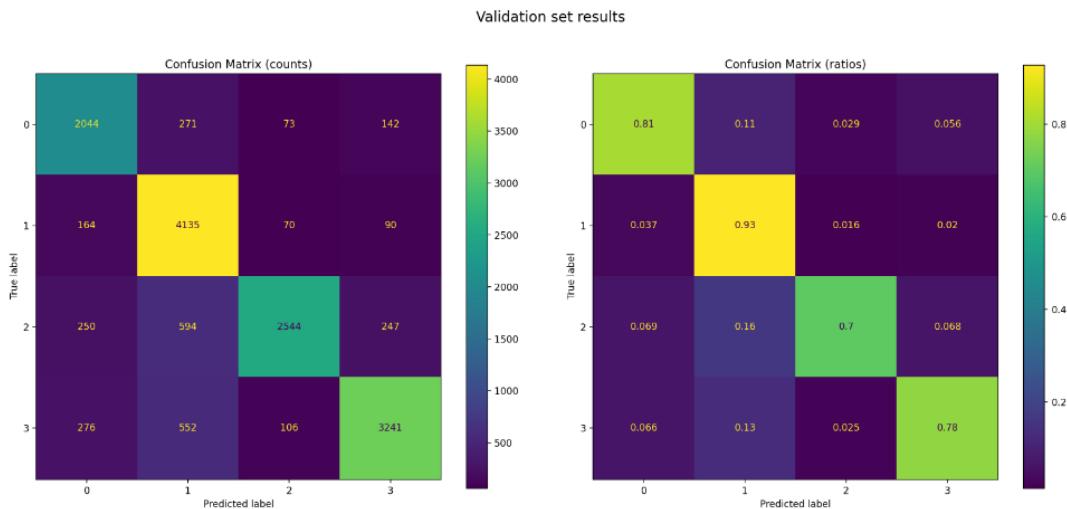
```

In [39]:
# Compute validation set predictions
pred = [np.argmax(i) for i in model.predict(X_va_seq)]

```

```
463/463 [=====] - 51s 107ms/step
```

```
In [40]:  
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay  
  
# Compute confusion matrix for raw counts  
cm_raw = confusion_matrix(y_va, pred)  
  
# Compute confusion matrix for normalized ratios  
cm_norm = confusion_matrix(y_va, pred, normalize='true')  
  
# Create confusion matrix plots  
disp_raw = ConfusionMatrixDisplay(confusion_matrix=cm_raw)  
disp_norm = ConfusionMatrixDisplay(confusion_matrix=cm_norm)  
  
# Plot confusion matrices side by side  
fig, axs = plt.subplots(1, 2, figsize=(20, 8), gridspec_kw={"width_ratios": [10, 10], "wspace": 0.1}, subplot_kw={"aspect": "equal"})  
plt.suptitle('Validation set results', size = 15)  
disp_raw.plot(ax=axs[0])  
axs[0].set_title('Confusion Matrix (counts)')  
disp_norm.plot(ax=axs[1])  
axs[1].set_title('Confusion Matrix (ratios)')  
plt.show()
```



Based on the confusion matrices in the validation set, we can see that the model performed better with 4 and 5-star ratings. This is not surprising given that these ratings had the largest number of samples available for the training.

Concerning the errors, overall the model tends to predict ratings that are close to each other, for instance often mistakenly predicting a rating of 2 for a rating of 3, or vice versa.

Make predictions

```
In [41]:  
# Compute test set predictions  
predictions = [np.argmax(i) for i in model.predict(X_te_seq)]  
  
# Create a new DataFrame to merge review ids and the model predictions  
submission = pd.DataFrame({'sentiment': df_test.sentiment, 'rating': predictions})  
  
# Check few random entries  
submission.sample(10)
```

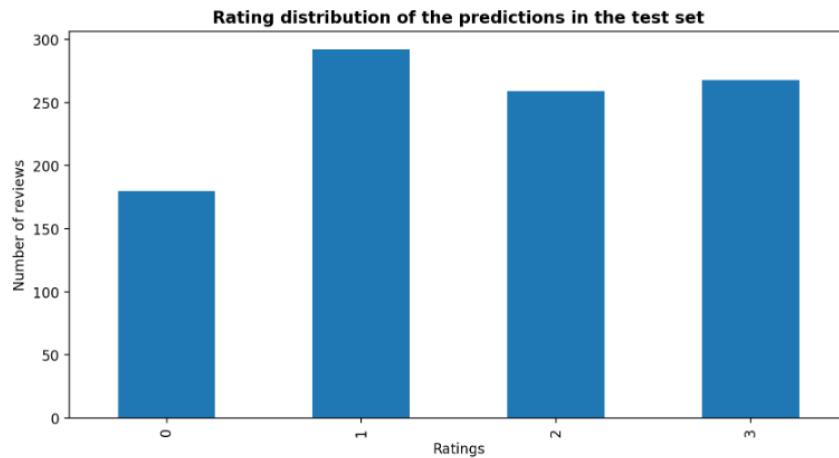
```
32/32 [=====] - 3s 102ms/step
```

Out[41]:

	sentiment	rating
975	3	3
976	2	2
324	0	0
94	1	1
693	3	1
586	2	2
880	3	3
127	1	1
540	2	2

469	0	0
-----	---	---

```
In [42]:  
# Check target balance  
submission['rating'].value_counts().sort_index().plot.bar(figsize=(10,5))  
plt.title('Rating distribution of the predictions in the test set', fontweight="bold")  
plt.xlabel('Ratings')  
plt.ylabel('Number of reviews')  
plt.show()
```



The target distribution in the test set is similar to the one in training set.

```
In [43]:  
# Export predictions  
submission.to_csv("submission.csv", index=None)
```

```
In [ ]:
```

License

This Notebook has been released under the [Apache 2.0](#) open source license.

Continue exploring



Input
1 file



Output
1 file



Logs
3901.7 second run - successful



Comments
0 comments

