# NLP: Yelp Review to Rating


## Authors: Tanvee Desai and Tanner Arrizabalaga


Hello! In this project, we will be looking over Yelp reviews (data available here: [https://www.yelp.com/dataset (https://www.yelp.com/dataset)](https://www.yelp.com/dataset)) and utilizing ML/DL to accurately predict what the reviews star rating is based solely on text.

This project is split into the following parts

- Libraries
- EDA
- Data Cleaning
  - Stop word removal, HTML parsing, punctuation removal, etc.
  - Creation of a cleaned *and* stemmed dataset
- Model Implementation
  - Simple BOW Model Neural Network
  - LSTM
  - Bidirectional LSTM
  - One vs. All LSTM Approach
- Exploring Challenges
  - Challenge 5
  - Challenge 6

## Importing necessary libraries

In [70]:
```
# General Libraries
import json
import sys
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import itertools

# NLP
import nltk
import re
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
from nltk.stem import PorterStemmer


# ML/DL
import tensorflow as tf
import pickle

from sklearn.preprocessing import LabelBinarizer, LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

from tensorflow import keras
from keras import Sequential
from keras.layers import Dense, Activation, Dropout, Embedding, Conv1D, MaxPoo
ling1D, LSTM, BatchNormalization, SpatialDropout1D, Bidirectional
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing import text, sequence
from keras import utils
from keras import regularizers
from keras.models import load_model
from keras.initializers import Constant
from keras.utils import plot_model
```

In [71]:
```
yelp = pd.read_json("./yelp_review_training_dataset.jsonl", lines = True)
yelp.head()
```

Out[71]:

| | review_id | text | stars |
|---|---|---|---|
| 0 | Q1sbwvVQXV2734tPgoKj4Q | Total bill for this horrible service? Over $8G... | 1 |
| 1 | GJXCdrto3ASJOqKeVWPi6Q | I *adore* Travis at the Hard Rock's new Kelly ... | 5 |
| 2 | 2TzJjDVDEuAW6MR5Vuc1ug | I have to say that this office really has it t... | 5 |
| 3 | yi0R0Ugj_xUx_Nek0-_Qig | Went in for a lunch. Steak sandwich was delici... | 5 |
| 4 | 11a8sVPMUFtaC7_ABRkmtw | Today was my second out of three sessions I ha... | 1 |

How large is the data?

In [72]:
```
yelp.shape
```

Out[72]:
```
(533581, 3)
```

## EDA - Stars

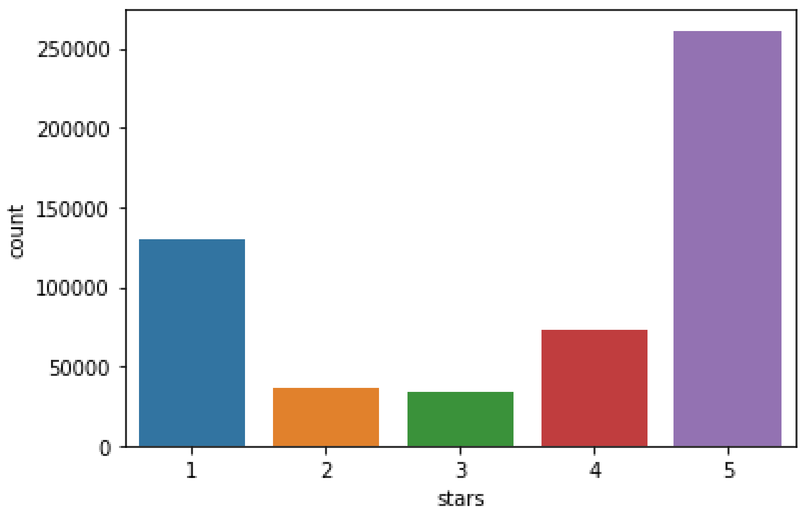
Not too much to go off of, but let's get a general understanding of our data. How many nulls do we have?

In [73]:
```
yelp.isna().sum()
```

Out[73]:
```
review_id    0
text         0
stars        0
dtype: int64
```

In [74]:
```
sns.countplot(yelp['stars'])
```

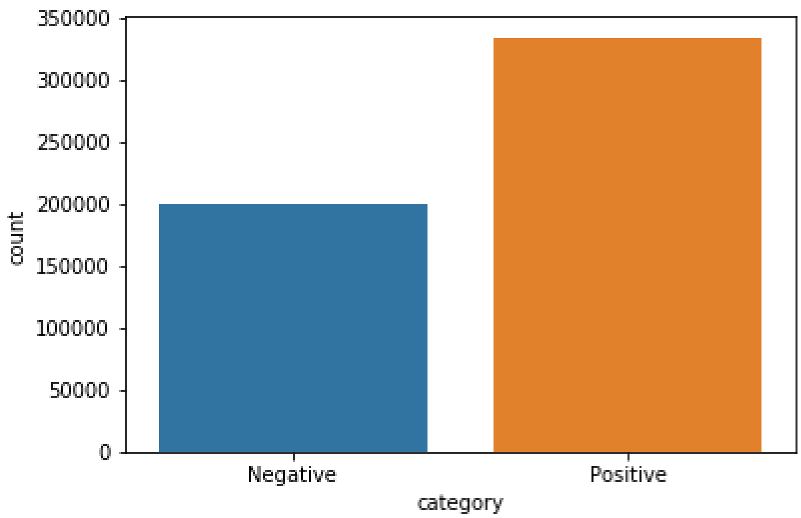
Out[74]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x25d04729548>
```




One thing we can potentially look at is whether or not the reviews are balanced. Let's say >=4 is positive, and <4 is negative. If we do see a significant difference in positive and negative reviews, we can balance it before training.

In [75]:
```
def pos_or_neg(x):
    if x >= 4:
        return "Positive"
    else:
        return "Negative"

yelp['category'] = yelp['stars'].apply(pos_or_neg)

sns.countplot(yelp['category'])
num_pos = np.count_nonzero(yelp['category'] == 'Positive')
num_neg = np.count_nonzero(yelp['category'] == 'Negative')
print("Positive to negative review ratio: ", num_pos / num_neg)
```

```
Positive to negative review ratio:  1.6679183395916979
```




There are roughly 1 and 2/3 times as many positive reviews as negative reviews. We will first try no class balancing when building the model, but may turn to class balancing later on.

## Data Cleaning - Text

In [77]:
```
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
STOPWORDS = set(stopwords.words('english'))
print(STOPWORDS)

def adjust_stopwords(stopwords):
    words_to_keep = set(['nor', 'not', 'very', 'no', 'few', 'too', 'doesn', 'd
idn', 'wasn', 'ain',
                        "doesn't", "isn't", "hasn't", 'shouldn', "weren't", "d
on't", "didn't",
                        "shouldn't", "wouldn't", "won't", "above", "below", "h
aven't", "shan't", "weren"
                        "but", "wouldn", "mightn", "under", "mustn't", "over",
"won", "aren", "wasn't",
                        "than"])
    return stopwords - words_to_keep

def clean_text(text):
    """
        text: a string

        return: modified initial string
    """
    new_text = BeautifulSoup(text, "lxml").text # HTML decoding
    new_text = new_text.lower() # lowercase text
    new_text = REPLACE_BY_SPACE_RE.sub(' ', new_text) # replace REPLACE_BY_SPA
CE_RE symbols by space in text
    new_text = BAD_SYMBOLS_RE.sub(' ', new_text) # delete symbols which are in
BAD_SYMBOLS_RE from text

    ps = PorterStemmer()

#     new_text = ' '.join(ps.stem(word) for word in new_text.split()) # keepin
g all words, no stop word removal
    new_text = ' '.join(ps.stem(word) for word in new_text.split() if word not
in STOPWORDS) # delete stopwords from text and stem
    return new_text

# STOPWORDS = adjust_stopwords(STOPWORDS)
print(STOPWORDS)
```

```
{'their', 'its', 'his', 're', "wouldn't", "you've", 'was', 'we', 'of', "you'r
e", 'a', 'do', 'while', 'been', 'into', 's', 'what', "should've", 'for', 'bef
ore', 'shan', 'o', 'mustn', 'because', 'or', "it's", 'they', 'and', 'off', 'o
ther', 'd', 'your', 'more', "shouldn't", 'during', 'who', "that'll", 'furthe
r', 'didn', 'so', 'from', 'all', 'wouldn', 'about', "mustn't", 'him', 'it',
'am', 'himself', 'doing', 'aren', 'an', 'are', 'being', 'now', 'shouldn', 'ov
er', 'to', 'you', 'y', 'than', 'just', 'with', "mightn't", 'yourselves', 'som
e', 'the', 'be', 'between', 'having', "wasn't", 'same', 'yours', 'down', "nee
dn't", 'were', 'he', 'll', 'how', 'doesn', 'but', 'this', 'ma', 'itself', 'th
emselves', 'once', 'had', 'those', 'is', 'not', 'm', 'ain', "couldn't", "yo
u'll", "aren't", 'mightn', 'by', 'any', 'where', 'own', 'on', 'hasn', 'both',
"doesn't", 'then', "shan't", 'until', 'under', 't', "isn't", 'through', 'was
n', 'did', 'them', 'won', 'up', "don't", 'such', 'after', 'here', 've', 'thes
e', "hasn't", "you'd", "she's", 'most', 'again', 'when', 'ours', 'too', 'abov
e', 'out', 'she', 'myself', 'each', 'below', 'have', 'why', 'will', 'in', 'wh
om', 'herself', 'at', "didn't", 'her', 'which', 'very', "weren't", 'only', 'i
f', 'ourselves', "hadn't", 'me', 'as', 'couldn', 'has', 'few', 'that', 'shoul
d', 'my', 'theirs', 'yourself', 'hers', 'our', 'no', 'can', 'haven', 'nor',
'needn', 'against', 'isn', 'there', 'i', 'does', "won't", "haven't", 'weren',
'hadn', 'don'}
```


```
{'their', 'its', 'his', 're', "wouldn't", "you've", 'was', 'we', 'of', "you'r
e", 'a', 'do', 'while', 'been', 'into', 's', 'what', "should've", 'for', 'bef
ore', 'shan', 'o', 'mustn', 'because', 'or', "it's", 'they', 'and', 'off', 'o
ther', 'd', 'your', 'more', "shouldn't", 'during', 'who', "that'll", 'furthe
r', 'didn', 'so', 'from', 'all', 'wouldn', 'about', "mustn't", 'him', 'it',
'am', 'himself', 'doing', 'aren', 'an', 'are', 'being', 'now', 'shouldn', 'ov
er', 'to', 'you', 'y', 'than', 'just', 'with', "mightn't", 'yourselves', 'som
e', 'the', 'be', 'between', 'having', "wasn't", 'same', 'yours', 'down', "nee
dn't", 'were', 'he', 'll', 'how', 'doesn', 'but', 'this', 'ma', 'itself', 'th
emselves', 'once', 'had', 'those', 'is', 'not', 'm', 'ain', "couldn't", "yo
u'll", "aren't", 'mightn', 'by', 'any', 'where', 'own', 'on', 'hasn', 'both',
"doesn't", 'then', "shan't", 'until', 'under', 't', "isn't", 'through', 'was
n', 'did', 'them', 'won', 'up', "don't", 'such', 'after', 'here', 've', 'thes
e', "hasn't", "you'd", "she's", 'most', 'again', 'when', 'ours', 'too', 'abov
e', 'out', 'she', 'myself', 'each', 'below', 'have', 'why', 'will', 'in', 'wh
om', 'herself', 'at', "didn't", 'her', 'which', 'very', "weren't", 'only', 'i
f', 'ourselves', "hadn't", 'me', 'as', 'couldn', 'has', 'few', 'that', 'shoul
d', 'my', 'theirs', 'yourself', 'hers', 'our', 'no', 'can', 'haven', 'nor',
'needn', 'against', 'isn', 'there', 'i', 'does', "won't", "haven't", 'weren',
'hadn', 'don'}
```


In [ ]:

```
%%time
yelp['text'] = yelp['text'].apply(clean_text)
yelp.to_csv('cleaned_yelp_stemmed.csv')
```

In [76]:
```
text_1 = "\"Good morning, cocktails for you?\" \nWait...what? Oh...it's Vegas! \n\nDining here, you best not be dieting because this place is literally the definition of excess, but in a good way. I'm a sucker for benedicts so that was awesome. \nService was really great too and the staff was so welcoming. It was our first stop just after landing so really appreciate the service.\n\nBack in Hawaii this reminds me of Zippys or Anna Millers - that home feeling. Prices are a bit high, but for what you get it's totally worth it. Will remember this place if I ever return to Vegas in the future."
text_2 = "80 bucks, thirty minutes to fix my shattered iPhone screen. Verizon won't help you so go here"
text_3 = "Tr\u00e8s grand caf\u00e9, mais aussi calme et reposant, je m'y suis arr\u00eat\u00e9 alors que j'\u00e9tais dans le coin.\n\nOn peu y mang\u00e9 le midi, prendre une p\u00e2tisserie ou un caf\u00e9/th\u00e9. \n\nJ'ai prit un th\u00e9 qui \u00e9tait vraiment bon, et je me suis pos\u00e9 devant une des grandes baies vitr\u00e9es sur un coussin et j'ai relax\u00e9 compl\u00e8tement pendant 2 heures. \n\nMais c'est aussi une coop\u00e9rative d'artiste, avec une estrade etc.\n\nIl y a aussi un magasin Bio \u00e0 l'entr\u00e9e o\u00f9 vous retrouverez des savons, huile d'olive et plein d'autres produits."
text_4 = "Sadly, as of July 28, 2016, Silverstein bakery is permanently closed. I went there today in person and found the bad news posted on their door. :("
text_5 = "I went here  they were about to close but the cashier was especially helpful ..but I guess they were tired of work..."

clean_text(text_4)
```

Out[76]:
```
'sadli juli 28 2016 silverstein bakeri perman close went today person found bad news post door'
```

# Model Implementation

## Evaluation

#### 1. Average Star Error (Average Absolute offset between predicted and true number of stars)

#### 2. Accuracy (Exact Match -- Number of exactly predicted star ratings / total samples)

In [78]:
```
from keras.losses import mean_absolute_error, binary_crossentropy, categorical
_crossentropy

def my_custom_loss_ova(y_true, y_pred):
    mse = mean_absolute_error(y_true, y_pred)
    crossentropy = binary_crossentropy(y_true, y_pred)
    return mse + crossentropy

def my_custom_loss(y_true, y_pred):
    mse = mean_absolute_error(y_true, y_pred)
    crossentropy = categorical_crossentropy(y_true, y_pred)
    return mse + crossentropy

def MAE(y_true, y_pred):
    diffs = np.abs(y_true - y_pred)
    loss = np.mean(diffs)
    return loss

def Accuracy(y_true, y_pred):
    correct = y_true == y_pred
    cor_count = np.count_nonzero(correct)
    return cor_count / len(y_true)

def custom_loss(y_true, y_pred):
    return MAE(y_true, y_pred) + Accuracy(y_true, y_pred)
```

## Train/Test Split (Unbalanced and balanced)

In [79]:
```
yelp = pd.read_csv('cleaned_yelp_stemmed.csv')
yelp.head()
```

Out[79]:

| | Unnamed: 0 | review_id | text | stars | category |
|---|---|---|---|---|---|
| **0** | 0 | Q1sbwvVQXV2734tPgoKj4Q | total bill horribl servic 8g crook actual nerv... | 1 | Negative |
| **1** | 1 | GJXCdrto3ASJOqKeVWPi6Q | ador travi hard rock new kelli cardena salon a... | 5 | Positive |
| **2** | 2 | 2TzJjDVDEuAW6MR5Vuc1ug | say offic realli togeth organ friendli dr j ph... | 5 | Positive |
| **3** | 3 | yi0R0Ugj_xUx_Nek0-_Qig | went lunch steak sandwich delici caesar salad ... | 5 | Positive |
| **4** | 4 | 11a8sVPMUFtaC7_ABRkmtw | today second three session paid although first... | 1 | Negative |

In [80]:
```
X = yelp['text'].fillna('').values
y = yelp['stars']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, rand
om_state=42)
```

In [81]:
```
%%time
max_words = 3000
tokenizer = text.Tokenizer(num_words=max_words, char_level=False)

tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_matrix(X_train)
X_test = tokenizer.texts_to_matrix(X_test)

encoder = LabelEncoder()
encoder.fit(y_train)
y_train = encoder.transform(y_train)
y_test = encoder.transform(y_test)

num_classes = np.max(y_train) + 1
y_train = utils.to_categorical(y_train, num_classes)
y_test = utils.to_categorical(y_test, num_classes)

print('X_train shape:', X_train.shape)
print('X_test shape:', X_test.shape)
print('y_train shape:', y_train.shape)
print('y_test shape:', y_test.shape)
```

```
X_train shape: (373506, 3000)
X_test shape: (160075, 3000)
y_train shape: (373506, 5)
y_test shape: (160075, 5)
Wall time: 50.6 s
```

Let's save the tokenizer as well for our test submission file script.

In [82]:
```
# # saving
# with open('tokenizer.pickle', 'wb') as handle:
#     pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

# # loading
# with open('tokenizer.pickle', 'rb') as handle:
#     tokenizer = pickle.load(handle)
```

## Baseline Sequential Model

Here, we are computing a single model, but in future we will optimize on several parameters, listed below

- Batch size
- Learning rate
- Gradient clipping
- Drop out
- Batch normalization
- Optimizers
- Regularization

After some tests, the main variations I noticed were from the learning rate, regularization, and the choice of the optimizer. With that being said, this baseline model will use **ADAM with a learning rate of .0001 and regularization (kernel, bias, and activity)**

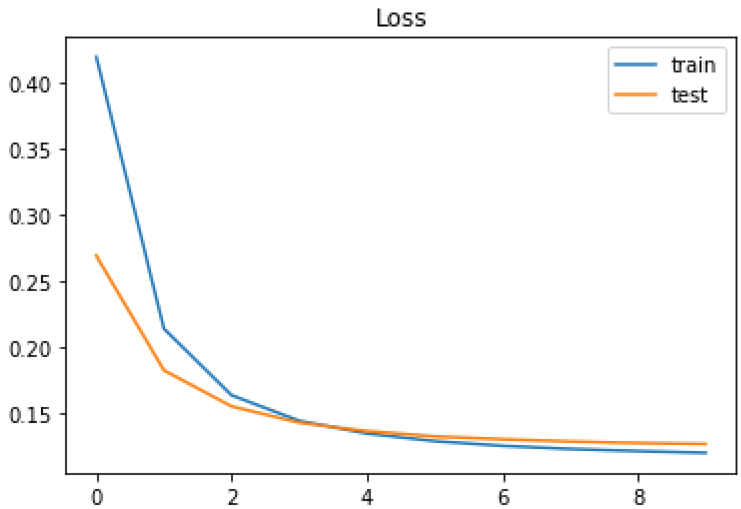
In [83]:
```
batch_size = 512
epochs = 10

lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=.0001,
    decay_steps=10000,
    decay_rate=0.9)

optimizer = keras.optimizers.Adam(learning_rate=lr_schedule, beta_1=0.9, beta_
2=0.95, amsgrad=False)

baseline = Sequential()
baseline.add(Dense(512, input_shape=(max_words,), kernel_regularizer=regulariz
ers.l1_l2(l1=1e-5, l2=1e-4),
          bias_regularizer=regularizers.l2(1e-4),
          activity_regularizer=regularizers.l2(1e-5)))
baseline.add(BatchNormalization())
baseline.add(Activation('relu'))
baseline.add(Dropout(0.3))
baseline.add(Dense(5))
baseline.add(Activation('softmax'))

baseline.compile(loss='mean_absolute_error',
              optimizer=optimizer,
              metrics=['accuracy', 'mean_absolute_error'])

history = baseline.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.2)
```

```
Train on 298804 samples, validate on 74702 samples
Epoch 1/10
298804/298804 [==============================] - 29s 97us/step - loss: 0.4188
- accuracy: 0.6918 - mean_absolute_error: 0.1338 - val_loss: 0.2691 - val_acc
uracy: 0.7278 - val_mean_absolute_error: 0.1155
Epoch 2/10
298804/298804 [==============================] - 15s 50us/step - loss: 0.2141
- accuracy: 0.7331 - mean_absolute_error: 0.1110 - val_loss: 0.1827 - val_acc
uracy: 0.7291 - val_mean_absolute_error: 0.1119
Epoch 3/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.1639
- accuracy: 0.7371 - mean_absolute_error: 0.1084 - val_loss: 0.1555 - val_acc
uracy: 0.7295 - val_mean_absolute_error: 0.1112
Epoch 4/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.1446
- accuracy: 0.7401 - mean_absolute_error: 0.1070 - val_loss: 0.1433 - val_acc
uracy: 0.7292 - val_mean_absolute_error: 0.1106
Epoch 5/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.1350
- accuracy: 0.7426 - mean_absolute_error: 0.1058 - val_loss: 0.1368 - val_acc
uracy: 0.7291 - val_mean_absolute_error: 0.1103
Epoch 6/10
298804/298804 [==============================] - 11s 38us/step - loss: 0.1293
- accuracy: 0.7450 - mean_absolute_error: 0.1048 - val_loss: 0.1329 - val_acc
uracy: 0.7297 - val_mean_absolute_error: 0.1100
Epoch 7/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.1257
- accuracy: 0.7465 - mean_absolute_error: 0.1041 - val_loss: 0.1307 - val_acc
uracy: 0.7303 - val_mean_absolute_error: 0.1100
Epoch 8/10
298804/298804 [==============================] - 11s 38us/step - loss: 0.1234
- accuracy: 0.7483 - mean_absolute_error: 0.1034 - val_loss: 0.1290 - val_acc
uracy: 0.7306 - val_mean_absolute_error: 0.1095
Epoch 9/10
298804/298804 [==============================] - 11s 36us/step - loss: 0.1218
- accuracy: 0.7496 - mean_absolute_error: 0.1028 - val_loss: 0.1278 - val_acc
uracy: 0.7313 - val_mean_absolute_error: 0.1093
Epoch 10/10
298804/298804 [==============================] - 11s 38us/step - loss: 0.1205
- accuracy: 0.7512 - mean_absolute_error: 0.1022 - val_loss: 0.1272 - val_acc
uracy: 0.7314 - val_mean_absolute_error: 0.1091
```

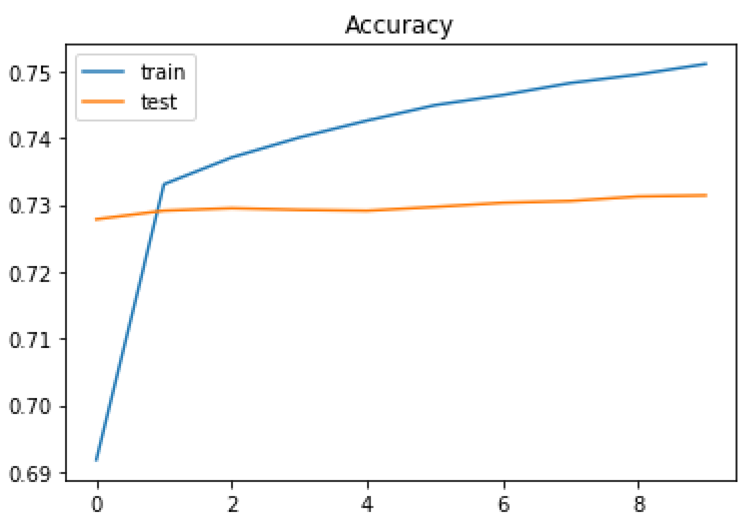
In [84]:
```
score = baseline.evaluate(X_test, y_test,
                       batch_size=batch_size, verbose=1)
print('Test accuracy:', score[1])
```

```
160075/160075 [==============================] - 19s 120us/step
Test accuracy: 0.7327939867973328
```

In [85]:
```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```




In [86]:
```
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()
```



In [87]:
```
# Get model output
y_pred = baseline.predict(X_test)

cols = [1, 2, 3, 4, 5]

# Creating predictions table
baseline_ps = pd.DataFrame(data=y_pred, columns=cols)
y_pred_true = baseline_ps.idxmax(axis=1)

# Creating truth
baseline_truth = pd.DataFrame(data=y_test, columns=cols)
y_test_true = baseline_truth.idxmax(axis=1)

# Confusion matrix
cm = confusion_matrix(y_pred_true, y_test_true)
pd.DataFrame(cm, index=cols, columns=cols)
```

Out[87]:

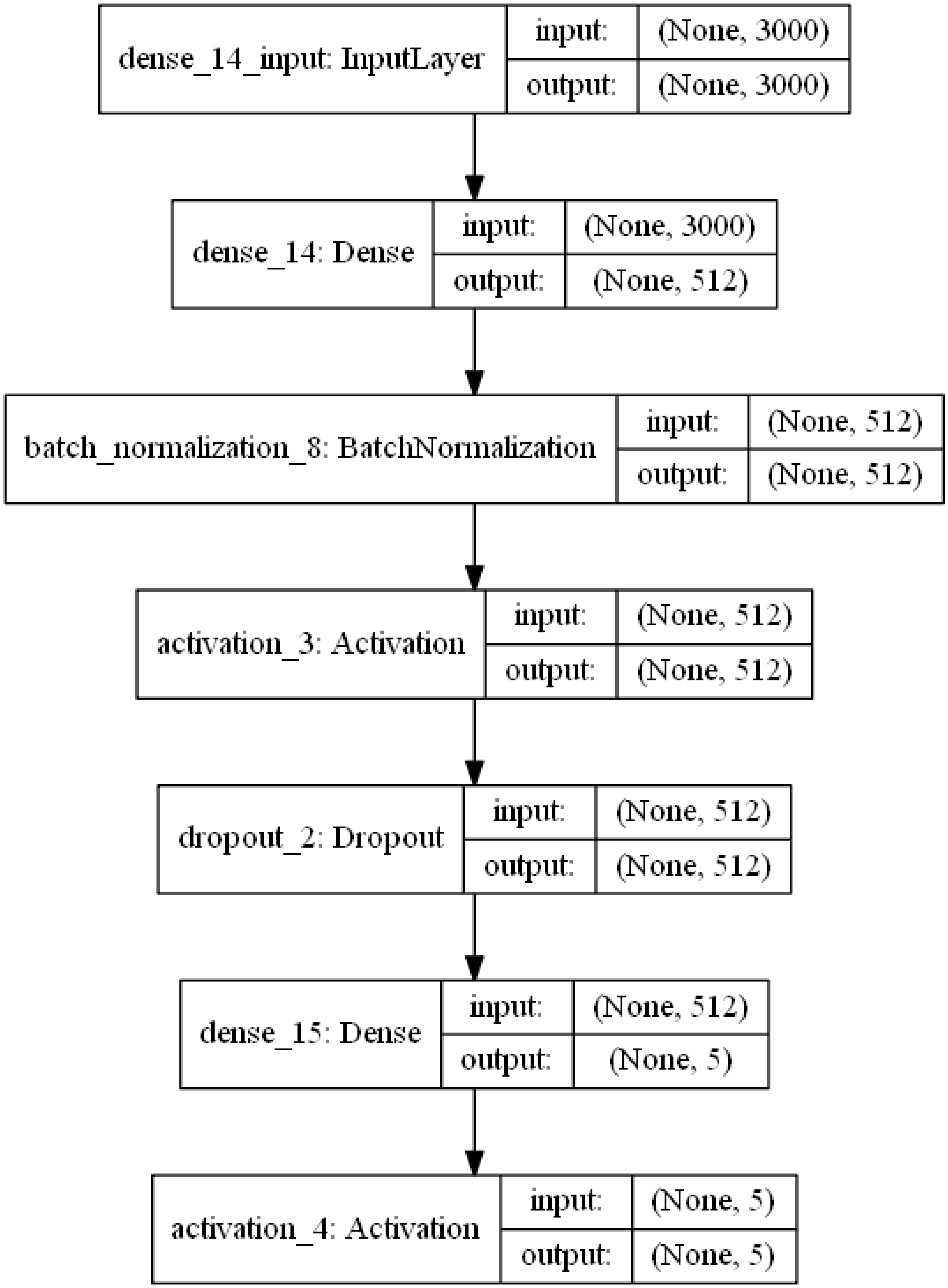
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **1** | 36726 | 7985 | 3529 | 1671 | 2130 |
| **2** | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 |
| **4** | 414 | 1507 | 4346 | 7491 | 3206 |
| **5** | 1747 | 1251 | 2388 | 12599 | 73085 |

In [88]:
```
print(classification_report(y_pred_true, y_test_true))
```

```
              precision    recall  f1-score   support

           1       0.94      0.71      0.81     52041
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.34      0.44      0.39     16964
           5       0.93      0.80      0.86     91070

    accuracy                           0.73    160075
   macro avg       0.44      0.39      0.41    160075
weighted avg       0.87      0.73      0.79    160075


C:\Users\Tanner\Anaconda3\envs\yelp\lib\site-packages\sklearn\metrics\_classi
fication.py:1272: UndefinedMetricWarning: Recall and F-score are ill-defined
and being set to 0.0 in labels with no true samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [89]:
```
plot_model(baseline, to_file='baseline.png', show_shapes=True)
```

Out[89]:




Let's save this model.

In [ ]:
```
# baseline.save('./models/baseline.h5')
```

## Now training with several parameter changes

In [ ]:
```
batch_sizes = [128, 256, 512]
epochs = [5]
learning_rates = [.01, .001, .0001]
dropout = [False, True]
batch_norm = [False, True]
regularization = [True]
optimizers = ["SGD", "RMSProp", "ADAM"]

all_lists = [batch_sizes, epochs, learning_rates, dropout, batch_norm, regular
ization, optimizers]

params_to_test = list(itertools.product(*all_lists))
print(len(params_to_test))
```

In [ ]:
```
models = {}
histories = {}
scores = {}

for params in params_to_test:
    print(params)
    batch_size, epochs, learning_rate, dropout, batch_norm, regularization, op
t = params

    if opt == "SGD":
        optimizer = keras.optimizers.SGD(learning_rate=learning_rate, momentum
=0.0, nesterov=False)
    elif opt == "RMSProp":
        optimizer = keras.optimizers.RMSprop(learning_rate=learning_rate, rho=
0.9)
    elif opt == "ADAM":
        optimizer = keras.optimizers.Adam(learning_rate=learning_rate, beta_1=
0.9, beta_2=0.99, amsgrad=False)
    else:
        optimizer = keras.optimizers.Adadelta(learning_rate=learning_rate, rho
=0.95)

    model = Sequential()
    model.add(Dense(512, input_shape=(max_words,), kernel_regularizer=regulari
zers.l1_l2(l1=1e-5, l2=1e-4)))

    # Check Batch Normalization
    if batch_norm:
        model.add(BatchNormalization())

    model.add(Activation('relu'))

    # Check Dropout
    if dropout:
        model.add(Dropout(0.2))

    model.add(Dense(5))
    model.add(Activation('softmax'))

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

    history = model.fit(X_train, y_train,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=0,
                        validation_split=0.1)

    models[params] = model
    histories[params] = history

    score = model.evaluate(X_test, y_test, batch_size=batch_size, verbose=1)
    print(score)

    scores[params] = score
```

# LSTM Model

### Specific Data Prep

In [90]:
```
%%time
X = yelp['text'].fillna('').values
y = pd.get_dummies(yelp['stars']).values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
m_state=42)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

max_words = 3000
maxlen = 400

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

# For the LSTM, we are going to pad our sequences
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
```

```
(373506,) (373506, 5)
(160075,) (160075, 5)
Wall time: 26.4 s
```

### LSTM #1

In [91]:
```
batch_size = 512
epochs = 5

lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=.001,
    decay_steps=10000,
    decay_rate=0.9)

optimizer = keras.optimizers.Adam(learning_rate=lr_schedule, beta_1=0.9, beta_
2=0.99, amsgrad=False, clipvalue=.3)

lstm = Sequential()
lstm.add(Embedding(max_words, 128, input_length=maxlen))
lstm.add(SpatialDropout1D(0.2))
lstm.add(Conv1D(64, 5, activation='relu', kernel_regularizer=regularizers.l1_l
2(l1=1e-5, l2=1e-4),
          bias_regularizer=regularizers.l2(1e-4)))
lstm.add(MaxPooling1D(pool_size=4))
lstm.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
lstm.add(BatchNormalization())
lstm.add(Dense(5, activation='sigmoid'))

lstm.compile(loss='mean_absolute_error',
              optimizer=optimizer,
              metrics=['accuracy', 'mean_absolute_error'])

history = lstm.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.2)
```

```
Train on 298804 samples, validate on 74702 samples
Epoch 1/5
298804/298804 [==============================] - 82s 273us/step - loss: 0.150
8 - accuracy: 0.6891 - mean_absolute_error: 0.1391 - val_loss: 0.1066 - val_a
ccuracy: 0.7196 - val_mean_absolute_error: 0.1016
Epoch 2/5
298804/298804 [==============================] - 81s 271us/step - loss: 0.103
5 - accuracy: 0.7214 - mean_absolute_error: 0.0991 - val_loss: 0.1025 - val_a
ccuracy: 0.7248 - val_mean_absolute_error: 0.0982
Epoch 3/5
298804/298804 [==============================] - 81s 270us/step - loss: 0.101
5 - accuracy: 0.7255 - mean_absolute_error: 0.0973 - val_loss: 0.1022 - val_a
ccuracy: 0.7256 - val_mean_absolute_error: 0.0980
Epoch 4/5
298804/298804 [==============================] - 81s 270us/step - loss: 0.100
6 - accuracy: 0.7274 - mean_absolute_error: 0.0964 - val_loss: 0.1027 - val_a
ccuracy: 0.7206 - val_mean_absolute_error: 0.0984
Epoch 5/5
298804/298804 [==============================] - 81s 273us/step - loss: 0.099
9 - accuracy: 0.7286 - mean_absolute_error: 0.0956 - val_loss: 0.1009 - val_a
ccuracy: 0.7217 - val_mean_absolute_error: 0.0966
```

### LSTM #1: Evaluation

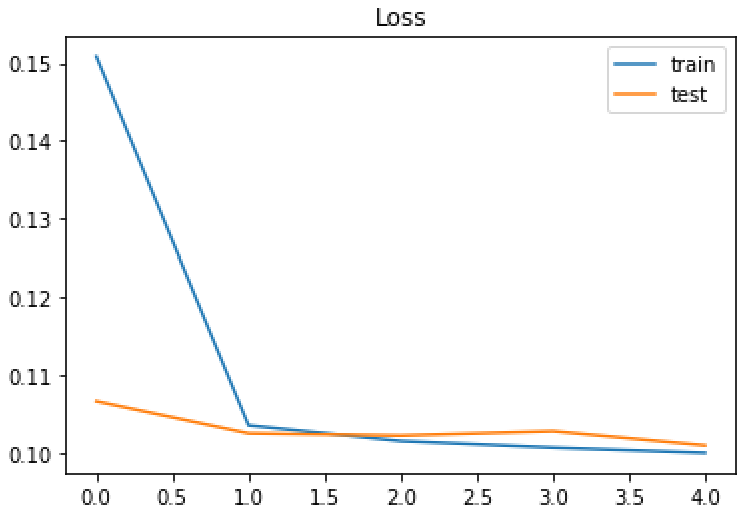
In [92]:
```
score = lstm.evaluate(X_test, y_test,
                       batch_size=batch_size, verbose=1)
print('Test accuracy:', score[1])
```

```
160075/160075 [==============================] - 10s 65us/step
Test accuracy: 0.7238357067108154
```

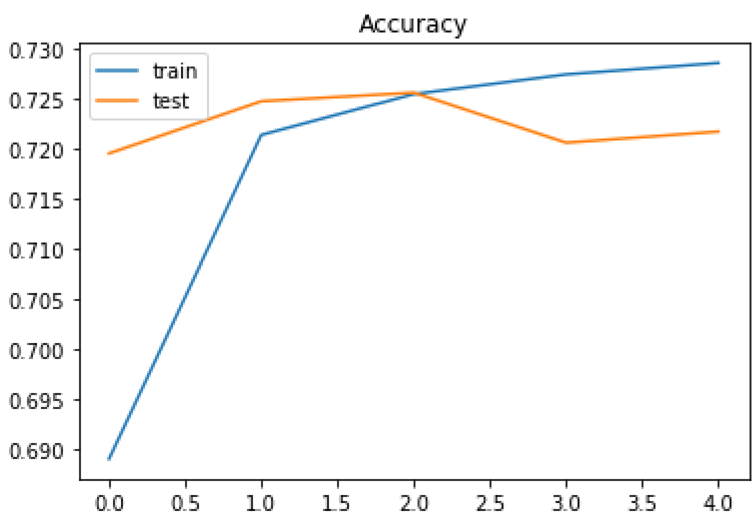
In [93]:
```
lstm.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 400, 128)          384000
_________________________________________________________________
spatial_dropout1d_7 (Spatial (None, 400, 128)          0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 396, 64)           41024
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 99, 64)            0
_________________________________________________________________
lstm_7 (LSTM)                (None, 128)               98816
_________________________________________________________________
batch_normalization_9 (Batch (None, 128)               512
_________________________________________________________________
dense_16 (Dense)             (None, 5)                 645
=================================================================
Total params: 524,997
Trainable params: 524,741
Non-trainable params: 256
_________________________________________________________________
```

In [94]:
```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```




In [95]:
```
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()
```



In [96]:
```
# Get model output
y_pred = lstm.predict(X_test)
y_pred

cols = [1, 2, 3, 4, 5]

# Creating predictions table
baseline_ps = pd.DataFrame(data=y_pred, columns=cols)
y_pred_true = baseline_ps.idxmax(axis=1)
y_pred_true

# Creating truth
baseline_truth = pd.DataFrame(data=y_test, columns=cols)
y_test_true = baseline_truth.idxmax(axis=1)
y_test_true

# Confusion matrix
cm = confusion_matrix(y_pred_true, y_test_true)
pd.DataFrame(cm, index=cols, columns=cols)
```

Out[96]:

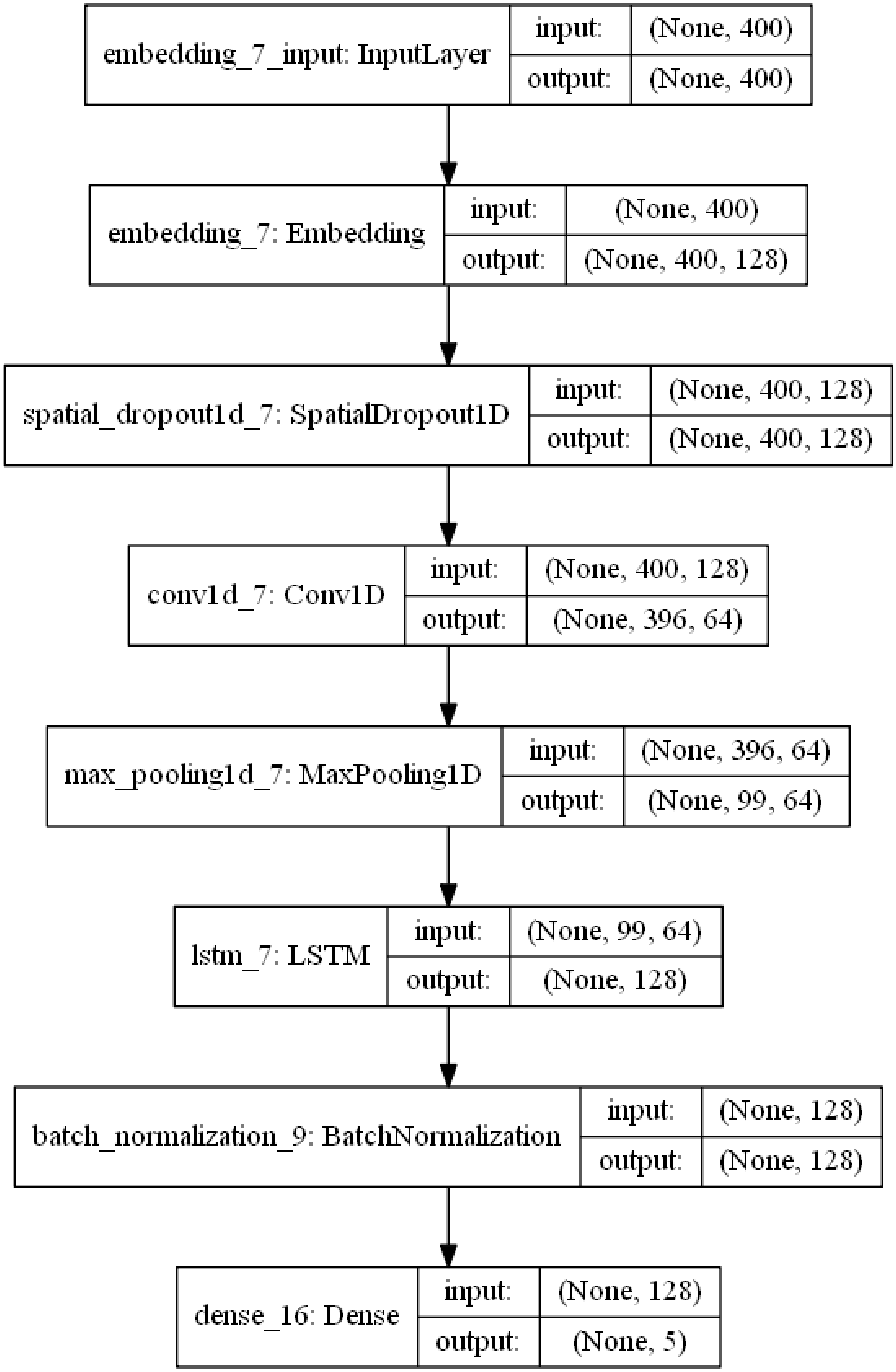
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **1** | 35664 | 6088 | 2110 | 1046 | 1468 |
| **2** | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 |
| **4** | 1046 | 2860 | 4633 | 4586 | 1335 |
| **5** | 2177 | 1795 | 3520 | 16129 | 75618 |

In [97]:
```
print(classification_report(y_pred_true, y_test_true))
```

```
              precision    recall  f1-score   support

           1       0.92      0.77      0.84     46376
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.21      0.32      0.25     14460
           5       0.96      0.76      0.85     99239

    accuracy                           0.72    160075
   macro avg       0.42      0.37      0.39    160075
weighted avg       0.88      0.72      0.79    160075


C:\Users\Tanner\Anaconda3\envs\yelp\lib\site-packages\sklearn\metrics\_classi
fication.py:1272: UndefinedMetricWarning: Recall and F-score are ill-defined
and being set to 0.0 in labels with no true samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [98]:
```
plot_model(lstm, to_file='baseline.png', show_shapes=True)
```

Out[98]:




Let's save this model as well.

In [ ]:
```
# lstm.save('./models/lstm.h5')
```

### LSTM #2

In [ ]:
```
batch_size = 128
epochs = 5

lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=.001,
    decay_steps=10000,
    decay_rate=0.9)

optimizer = keras.optimizers.Adam(learning_rate=lr_schedule, beta_1=0.9, beta_
2=0.99, amsgrad=False, clipvalue=.3)

lstm_v2 = Sequential()
lstm_v2.add(Embedding(max_words, 128, input_length=maxlen))
lstm_v2.add(SpatialDropout1D(0.3))
lstm_v2.add(Bidirectional(LSTM(128, dropout=0.3, recurrent_dropout=0.3)))
lstm_v2.add(Dense(128, activation='relu'))
lstm_v2.add(Dropout(0.2))
lstm_v2.add(Dense(128, activation='relu'))
lstm_v2.add(Dropout(0.2))
lstm_v2.add(Dense(5, activation='sigmoid'))

lstm_v2.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

history = lstm_v2.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.2)
```

### LSTM #2: Evaluation

In [ ]:
```
score = lstm_v2.evaluate(X_test, y_test,
                       batch_size=batch_size, verbose=1)
print('Test accuracy:', score[1])
```

In [ ]:
```
lstm_v2.summary()
```

In [ ]:
```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```

In [ ]:
```
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()
```

Let's save this model as well.

In [ ]:
```
lstm.save('./models/lstm_v2.h5')
```

## One vs. All Approach

In the one vs. all approach, it goes by the following idea:

- We will have $N$ learners for the multi-class classification problem, where $N$ is the number of classes
- For each learner $L$, we will train $L$ on our training data $X_{Train}$ and $y_{Train}$. However, $y_{Train}$ consists of only one label, making it a binary classification problem instead of multinomial
  - For instance, learner $L_1$ will still use all of $X_{Train}$, but $y_{Train}$ will now be transformed to be a binary vector $v_i$ where $i$ denotes the star rating we are attempting to predict
- Once we have concluded our training, we will then create an ensemble model (bagging) that does the following
  1. $L_1$, $L_2$, ..., $L_5$ all assign $p_i$ to each record in $X_{Test}$, where $p_i$ is the likelihood observation $x_n$ belongs to class $i$
  2. From there, our prediction is the following: $P_n = argmax(p_1, p_2, p_3, p_4, p_5)$

After observing the challenge datasets 5 & 6, my partner and I believe this approach is a clever way to tackle the challenges while still having a strong model.

Sources: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/one-vs-all (https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/one-vs-all)

In [99]:
```
yelp = pd.read_csv('cleaned_yelp_stemmed.csv')

X = yelp['text'].fillna('').values
y = pd.get_dummies(yelp['stars']).values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, rand
om_state=42)

# Loading
# with open('tokenizer.pickle', 'rb') as handle:
#     tokenizer = pickle.load(handle)

max_words = 3000
maxlen = 400

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)

print('X_train shape:', X_train.shape)
print('X_test shape:', X_test.shape)
print('y_train shape:', y_train.shape)
print('y_test shape:', y_test.shape)
```

```
X_train shape: (373506, 400)
X_test shape: (160075, 400)
y_train shape: (373506, 5)
y_test shape: (160075, 5)
```

#### Buidling all models

In [100]:
```
stars = np.arange(1, 6)
models = {}
histories = {}
batch_size = 512

for star in stars:
    if star in [1, 2]:
        epochs = 2
    elif star in [3, 4]:
        epochs = 3
    else:
        epochs = 4

    print(star)
    y_train_sub = y_train[:, star - 1]

    lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=.001,
    decay_steps=10000,
    decay_rate=0.9)

    optimizer = keras.optimizers.Adam(learning_rate=lr_schedule, beta_1=0.9, b
eta_2=0.99, amsgrad=False, clipvalue=.3)

    sub_lstm = Sequential()
    sub_lstm.add(Embedding(max_words, 128, input_length=maxlen))
    sub_lstm.add(SpatialDropout1D(0.2))
    sub_lstm.add(Conv1D(64, 5, activation='relu', kernel_regularizer=regulariz
ers.l1_l2(l1=1e-5, l2=1e-4),
              bias_regularizer=regularizers.l2(1e-4)))
    sub_lstm.add(MaxPooling1D(pool_size=4))
    sub_lstm.add(LSTM(128))
    sub_lstm.add(BatchNormalization())
    sub_lstm.add(Dense(8))
    sub_lstm.add(Dense(1, activation='sigmoid'))

    sub_lstm.compile(loss='mean_absolute_error',
                  optimizer=optimizer,
                  metrics=['accuracy', 'mean_absolute_error'])

    history = sub_lstm.fit(X_train, y_train_sub,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=1,
                        validation_split=0.2)

    models[star] = sub_lstm
    histories[star] = sub_lstm
```

```
1
Train on 298804 samples, validate on 74702 samples
Epoch 1/2
298804/298804 [==============================] - 79s 265us/step - loss: 0.118
9 - accuracy: 0.8992 - mean_absolute_error: 0.1029 - val_loss: 0.1118 - val_a
ccuracy: 0.8983 - val_mean_absolute_error: 0.1017
Epoch 2/2
298804/298804 [==============================] - 78s 260us/step - loss: 0.092
8 - accuracy: 0.9160 - mean_absolute_error: 0.0844 - val_loss: 0.1062 - val_a
ccuracy: 0.9011 - val_mean_absolute_error: 0.0988
2
Train on 298804 samples, validate on 74702 samples
Epoch 1/2
298804/298804 [==============================] - 78s 259us/step - loss: 0.099
8 - accuracy: 0.9181 - mean_absolute_error: 0.0867 - val_loss: 0.0706 - val_a
ccuracy: 0.9323 - val_mean_absolute_error: 0.0678
Epoch 2/2
298804/298804 [==============================] - 77s 258us/step - loss: 0.067
6 - accuracy: 0.9329 - mean_absolute_error: 0.0671 - val_loss: 0.0678 - val_a
ccuracy: 0.9323 - val_mean_absolute_error: 0.0677
3
Train on 298804 samples, validate on 74702 samples
Epoch 1/3
298804/298804 [==============================] - 80s 267us/step - loss: 0.098
0 - accuracy: 0.9178 - mean_absolute_error: 0.0865 - val_loss: 0.0653 - val_a
ccuracy: 0.9357 - val_mean_absolute_error: 0.0644
Epoch 2/3
298804/298804 [==============================] - 79s 265us/step - loss: 0.072
7 - accuracy: 0.9279 - mean_absolute_error: 0.0722 - val_loss: 0.0638 - val_a
ccuracy: 0.9363 - val_mean_absolute_error: 0.0637
Epoch 3/3
298804/298804 [==============================] - 82s 275us/step - loss: 0.064
4 - accuracy: 0.9356 - mean_absolute_error: 0.0644 - val_loss: 0.0638 - val_a
ccuracy: 0.9363 - val_mean_absolute_error: 0.0637
4
Train on 298804 samples, validate on 74702 samples
Epoch 1/3
298804/298804 [==============================] - 81s 270us/step - loss: 0.179
9 - accuracy: 0.8366 - mean_absolute_error: 0.1678 - val_loss: 0.1382 - val_a
ccuracy: 0.8639 - val_mean_absolute_error: 0.1363
Epoch 2/3
298804/298804 [==============================] - 81s 270us/step - loss: 0.135
7 - accuracy: 0.8645 - mean_absolute_error: 0.1355 - val_loss: 0.1371 - val_a
ccuracy: 0.8639 - val_mean_absolute_error: 0.1370
Epoch 3/3
298804/298804 [==============================] - 78s 263us/step - loss: 0.135
6 - accuracy: 0.8645 - mean_absolute_error: 0.1355 - val_loss: 0.1362 - val_a
ccuracy: 0.8639 - val_mean_absolute_error: 0.1361
5
Train on 298804 samples, validate on 74702 samples
Epoch 1/4
298804/298804 [==============================] - 80s 266us/step - loss: 0.174
6 - accuracy: 0.8446 - mean_absolute_error: 0.1571 - val_loss: 0.1633 - val_a
ccuracy: 0.8495 - val_mean_absolute_error: 0.1511
Epoch 2/4
298804/298804 [==============================] - 79s 264us/step - loss: 0.149
0 - accuracy: 0.8616 - mean_absolute_error: 0.1387 - val_loss: 0.1606 - val_a
```

```
ccuracy: 0.8488 - val_mean_absolute_error: 0.1514
Epoch 3/4
298804/298804 [==============================] - 80s 267us/step - loss: 0.144
5 - accuracy: 0.8642 - mean_absolute_error: 0.1360 - val_loss: 0.1549 - val_a
ccuracy: 0.8533 - val_mean_absolute_error: 0.1469
Epoch 4/4
298804/298804 [==============================] - 82s 273us/step - loss: 0.140
7 - accuracy: 0.8674 - mean_absolute_error: 0.1327 - val_loss: 0.1491 - val_a
ccuracy: 0.8591 - val_mean_absolute_error: 0.1410
```

### Building an ensemble model (maximization between learners) for all trained models

#### *Testing*

In [101]:

```
%%time
# Evaluating the models above (TEST)
y_test_und = pd.DataFrame(y_test)
y_test_true = pd.DataFrame(y_test_und.columns[np.where(y_test_und!=0)[1]]) + 1

# Unload models
lstm_1, lstm_2, lstm_3, lstm_4, lstm_5 = models[1], models[2], models[3], mode
ls[4], models[5]

## Predicting the probability for each observation each model
print("Predicting 1 star")
one_star_ps = lstm_1.predict(X_test)
print("Predicting 2 star")
two_star_ps = lstm_2.predict(X_test)
print("Predicting 3 star")
three_star_ps = lstm_3.predict(X_test)
print("Predicting 4 star")
four_star_ps = lstm_4.predict(X_test)
print("Predicting 5 star")
five_star_ps = lstm_5.predict(X_test)

data = [one_star_ps.flatten(), two_star_ps.flatten(), three_star_ps.flatten(),
four_star_ps.flatten(), five_star_ps.flatten()]
cols = [1, 2, 3, 4, 5]
ps = pd.DataFrame(data=data, index=cols).T

ps["pred"] = ps.idxmax(axis=1)
ps.head()

print(MAE(ps["pred"], y_test_true[0]))
print(Accuracy(ps["pred"], y_test_true[0]))
```

```
Predicting 1 star
Predicting 2 star
Predicting 3 star
Predicting 4 star
Predicting 5 star
0.5207059191004216
0.6846728096204904
Wall time: 5min 49s
```

In [102]:
```
# Confusion matrix
cm = confusion_matrix(ps["pred"], y_test_true[0])
pd.DataFrame(cm, index=cols, columns=cols)
```

Out[102]:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **1** | 31023 | 3627 | 908 | 367 | 589 |
| **2** | 5980 | 6247 | 7526 | 9242 | 5503 |
| **3** | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 |
| **5** | 1884 | 869 | 1829 | 12152 | 72329 |

In [103]:
```
print(classification_report(ps["pred"], y_test_true[0]))
```

```
              precision    recall  f1-score   support

           1       0.80      0.85      0.82     36514
           2       0.58      0.18      0.28     34498
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.92      0.81      0.86     89063

    accuracy                           0.68    160075
   macro avg       0.46      0.37      0.39    160075
weighted avg       0.82      0.68      0.73    160075


C:\Users\Tanner\Anaconda3\envs\yelp\lib\site-packages\sklearn\metrics\_classi
fication.py:1272: UndefinedMetricWarning: Recall and F-score are ill-defined
and being set to 0.0 in labels with no true samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

### Saving the models

In [ ]:
```
# lstm_1.save("./models/one_star.h5")
# lstm_2.save("./models/two_star.h5")
# lstm_3.save("./models/three_star.h5")
# lstm_4.save("./models/four_star.h5")
# lstm_5.save("./models/five_star.h5")
```

## Ensemble on Test Set

In [104]:
```
yelp = pd.read_csv('cleaned_yelp_stemmed.csv')

X = yelp['text'].fillna('').values
y = pd.get_dummies(yelp['stars'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, rand
om_state=42)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

max_words = 3000
maxlen = 400

# with open('tokenizer.pickle', 'rb') as handle:
#     tokenizer = pickle.load(handle)

print(y_test)

necc_cols = [1, 2, 3, 4, 5]
for col in necc_cols:
    if col not in y_test.columns:
        y_test[col] = 0

y_test = y_test[necc_cols]
y_test = y_test.values

X_baseline = tokenizer.texts_to_matrix(X_test)
X_lstm = tokenizer.texts_to_sequences(X_test)
X_lstm = pad_sequences(X_lstm, maxlen=maxlen)
```

```
(373506,) (373506, 5)
(160075,) (160075, 5)
        1  2  3  4  5
255947  0  0  0  0  1
261035  0  0  0  0  1
355633  0  0  0  0  1
205506  0  0  0  0  1
97222   0  0  0  1  0
...    .. .. .. .. ..
491832  0  0  0  0  1
311959  0  0  0  0  1
140524  1  0  0  0  0
125037  0  0  1  0  0
200135  0  0  0  1  0

[160075 rows x 5 columns]
```

In [ ]:
```
# # Trying our pretrained models
# # Optimizer
# lr_schedule = keras.optimizers.schedules.ExponentialDecay(initial_learning_r
ate=.001, decay_steps=10000, decay_rate=0.9)
# optimizer = keras.optimizers.Adam(learning_rate=lr_schedule, beta_1=0.9, bet
a_2=0.99, amsgrad=False, clipvalue=.3)

# # Baseline
# baseline = load_model('./models/baseline.h5')

# baseline.compile(loss='categorical_crossentropy',
#                optimizer=optimizer,
#                metrics=['accuracy'])

# # LSTM
# lstm = load_model('./models/lstm.h5')

# lstm.compile(loss='categorical_crossentropy',
#                optimizer=optimizer,
#                metrics=['accuracy'])


# # One vs. all
# lstm_1 = load_model('./models/one_star.h5')

# lstm_1.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_2 = load_model('./models/two_star.h5')

# lstm_2.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_3 = load_model('./models/three_star.h5')

# lstm_3.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_4 = load_model('./models/four_star.h5')

# lstm_4.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_5 = load_model('./models/five_star.h5')

# lstm_5.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])
```

In [105]:
```
cols = [1, 2, 3, 4, 5]
# Baseline
print("Baseline")
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# LSTM
print("LSTM")
lstm_preds = pd.DataFrame(lstm.predict(X_lstm), columns=cols)
lstm_preds['lstm_pred'] = lstm_preds.idxmax(axis=1)

# One vs. all
print("OVA")
one_star_ps = lstm_1.predict(X_lstm)
two_star_ps = lstm_2.predict(X_lstm)
three_star_ps = lstm_3.predict(X_lstm)
four_star_ps = lstm_4.predict(X_lstm)
five_star_ps = lstm_5.predict(X_lstm)

data = [one_star_ps.flatten(), two_star_ps.flatten(), three_star_ps.flatten(),
four_star_ps.flatten(), five_star_ps.flatten()]
ova_preds = pd.DataFrame(data=data, index=cols).T

ova_preds["ova_pred"] = ova_preds.idxmax(axis=1)

all_preds = pd.DataFrame([baseline_preds['baseline_pred'], lstm_preds['lstm_pr
ed'], ova_preds['ova_pred']]).T
all_preds["final_pred"] = all_preds.mode(axis=1)[0]
```

```
Baseline
LSTM
OVA
```

In [106]:
```
print([MAE(all_preds["final_pred"], pd.DataFrame(data=y_test, columns=cols).id
xmax(axis=1)), Accuracy(all_preds["final_pred"], pd.DataFrame(data=y_test, col
umns=cols).idxmax(axis=1))])
```

```
[0.4583663907543339, 0.7212244260502889]
```

In [107]:
```
# Confusion matrix
cm = confusion_matrix(all_preds["final_pred"], pd.DataFrame(data=y_test, colum
ns=cols).idxmax(axis=1))
pd.DataFrame(cm, index=cols, columns=cols)
```

Out[107]:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **1** | 36862 | 8112 | 3629 | 1757 | 1804 |
| **2** | 148 | 332 | 916 | 2346 | 1271 |
| **3** | 0 | 0 | 0 | 0 | 0 |
| **4** | 237 | 1154 | 3251 | 3887 | 977 |
| **5** | 1640 | 1145 | 2467 | 13771 | 74369 |

In [108]:
```python
print(classification_report(y_pred_true, y_test_true))
```

```
              precision    recall  f1-score   support

           1       0.92      0.77      0.84     46376
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.21      0.32      0.25     14460
           5       0.96      0.76      0.85     99239

    accuracy                           0.72    160075
   macro avg       0.42      0.37      0.39    160075
weighted avg       0.88      0.72      0.79    160075

C:\Users\Tanner\Anaconda3\envs\yelp\lib\site-packages\sklearn\metrics\_classi
fication.py:1272: UndefinedMetricWarning: Recall and F-score are ill-defined
and being set to 0.0 in labels with no true samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

# Challenges

## Challenge 5

In [109]:
```python
c5 = pd.read_json("./yelp_challenge_5_with_answers.jsonl", lines = True)
print(c5.shape)
c5.head()
```

```
(500, 3)
```

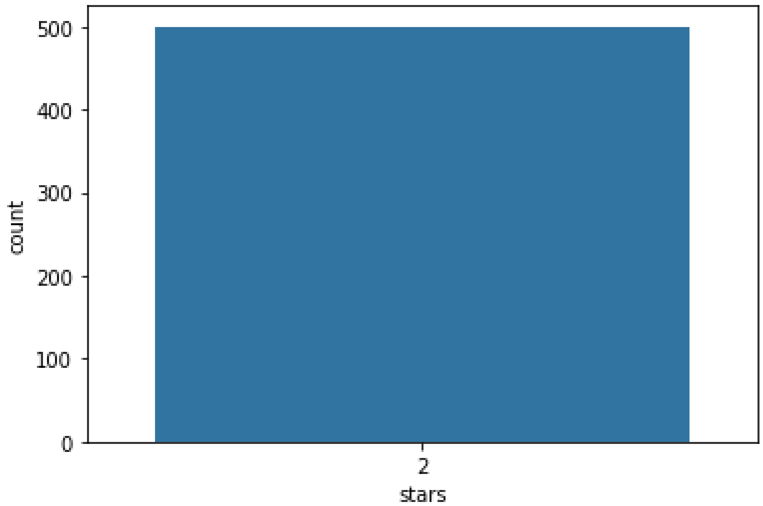
Out[109]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | 50 | I went to this campus for 1 semester. I was in... | 2 |
| **1** | 51 | I have rated it a two star based on its compar... | 2 |
| **2** | 52 | Just like most of the reviews, we ordered and ... | 2 |
| **3** | 53 | I only go here if it is an emergency. I HATE i... | 2 |
| **4** | 54 | Rude staff. I got 60 feeder fish and about 15 ... | 2 |

***Quick EDA***

In [110]:
```
sns.countplot(c5['stars'])
```

Out[110]: <matplotlib.axes._subplots.AxesSubplot at 0x25c76891b48>




### *Pre-processing*

In [111]:
```
c5['text'] = c5['text'].apply(clean_text)
c5.head()
```

Out[111]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | 50 | went campu 1 semest busi inform system campu o... | 2 |
| **1** | 51 | rate two star base comparison shop find staff ... | 2 |
| **2** | 52 | like review order paid half front door advanc ... | 2 |
| **3** | 53 | go emerg hate one door enter exit loss prevent... | 2 |
| **4** | 54 | rude staff got 60 feeder fish 15 dead cashier ... | 2 |

### *Load previous tokenizer*

In [112]:
```
X = c5['text'].fillna('').values
y = pd.get_dummies(c5['stars'])

# with open('tokenizer.pickle', 'rb') as handle:
#     tokenizer = pickle.load(handle)

max_words

necc_cols = [1, 2, 3, 4, 5]
for col in necc_cols:
    if col not in y.columns:
        y[col] = 0

y = y[necc_cols]
y = y.values

X_baseline = tokenizer.texts_to_matrix(X)
X_lstm = tokenizer.texts_to_sequences(X)
X_lstm = pad_sequences(X_lstm, maxlen=400)
```

#### ***Load and compile models***

In [ ]:
```
# # Baseline
# baseline = load_model('./models/baseline.h5')

# baseline.compile(loss='categorical_crossentropy',
#                optimizer=optimizer,
#                metrics=['accuracy'])

# # LSTM
# lstm = load_model('./models/lstm.h5')

# lstm.compile(loss='categorical_crossentropy',
#                optimizer=optimizer,
#                metrics=['accuracy'])


# # One vs. all
# lstm_1 = load_model('./models/one_star.h5')

# lstm_1.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_2 = load_model('./models/two_star.h5')

# lstm_2.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_3 = load_model('./models/three_star.h5')

# lstm_3.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_4 = load_model('./models/four_star.h5')

# lstm_4.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_5 = load_model('./models/five_star.h5')

# lstm_5.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])
```

### *Evaluate Models*

In [113]:
```
# Baseline
print(baseline.evaluate(X_baseline, y))

# LSTM
print(lstm.evaluate(X_lstm, y))

# One vs. All
one_star_ps = lstm_1.predict(X_lstm)
two_star_ps = lstm_2.predict(X_lstm)
three_star_ps = lstm_3.predict(X_lstm)
four_star_ps = lstm_4.predict(X_lstm)
five_star_ps = lstm_5.predict(X_lstm)

data = [one_star_ps.flatten(), two_star_ps.flatten(), three_star_ps.flatten(),
four_star_ps.flatten(), five_star_ps.flatten()]
cols = [1, 2, 3, 4, 5]
ps = pd.DataFrame(data=data, index=cols).T

ps["ova_pred"] = ps.idxmax(axis=1)

print([MAE(ps["ova_pred"], pd.DataFrame(data=y, columns=cols).idxmax(axis=1)),
Accuracy(ps["ova_pred"], pd.DataFrame(data=y, columns=cols).idxmax(axis=1))])
```

```
500/500 [==============================] - 0s 74us/step
[0.4165544147491455, 0.0, 0.3999996483325958]
500/500 [==============================] - 0s 508us/step
[0.28171328973770143, 0.0, 0.27740949392318726]
[0.616, 0.56]
```

***Attempt Ensemble***

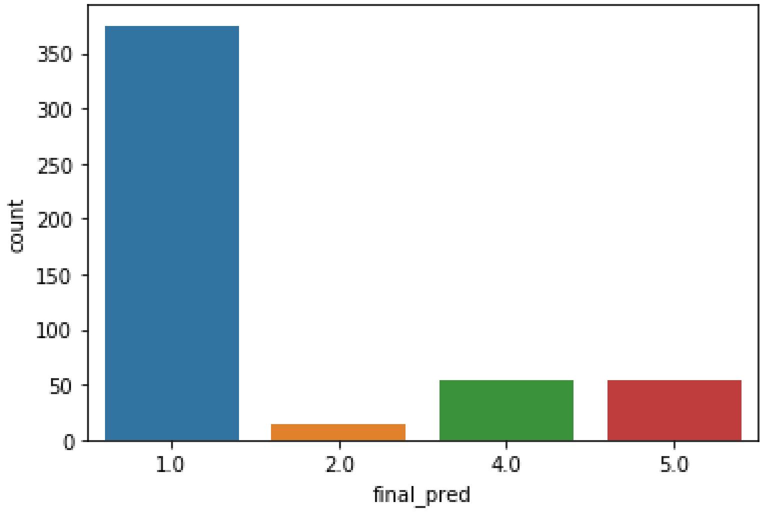
In [114]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# LSTM
lstm_preds = pd.DataFrame(lstm.predict(X_lstm), columns=cols)
lstm_preds['lstm_pred'] = lstm_preds.idxmax(axis=1)

# One vs. all
ova_preds = ps

all_preds = pd.DataFrame([baseline_preds['baseline_pred'], lstm_preds['lstm_pr
ed'], ova_preds['ova_pred']]).T
all_preds["final_pred"] = all_preds.mode(axis=1)[0]

print([MAE(all_preds["final_pred"], pd.DataFrame(data=y, columns=cols).idxmax(
axis=1)), Accuracy(all_preds["final_pred"], pd.DataFrame(data=y, columns=cols)
.idxmax(axis=1))])
```

```
[1.3, 0.03]
```

### *Misc.*

In [115]:
```
sns.countplot(all_preds["final_pred"])
```

Out[115]: <matplotlib.axes._subplots.AxesSubplot at 0x25ba509c448>




## Challenge 6

In [116]:
```
c6 = pd.read_json("./yelp_challenge_6_with_answers.jsonl", lines = True)
print(c6.shape)
c6.head()
```

```
(500, 3)
```

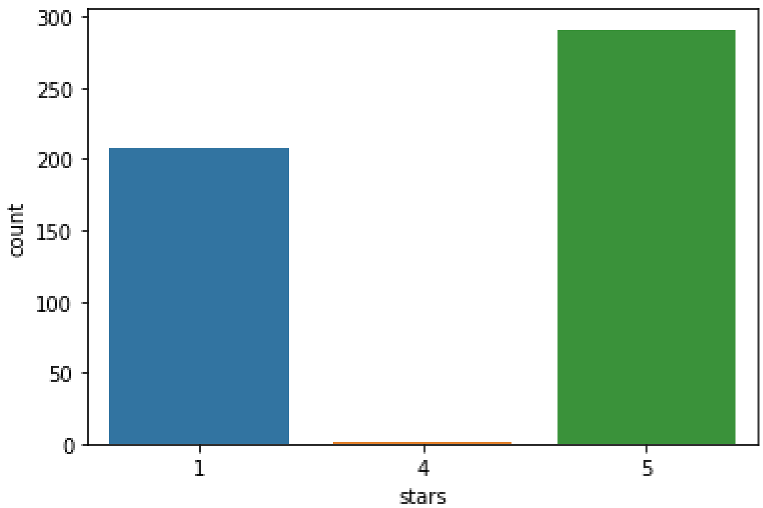
Out[116]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | 60 | Amazing for Trees\n\n$20 for a 5 gallon . I wi... | 5 |
| **1** | 61 | How the hell can Taco Bell be closed before mi... | 5 |
| **2** | 62 | I actually had no intention of visiting this p... | 5 |
| **3** | 63 | Yesterday around 3:30 pm I was driving west on... | 5 |
| **4** | 64 | DR FITZMAURICE did surgery on both hands on th... | 5 |

### *Quick EDA*

In [117]:
```
sns.countplot(c6['stars'])
```

Out[117]: <matplotlib.axes._subplots.AxesSubplot at 0x25ba504cb88>




### *Pre-processing*

In [118]:
```
c6['text'] = c6['text'].apply(clean_text)
c6.head()
```

Out[118]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | 60 | amaz tree 20 5 gallon never go low home depot ... | 5 |
| **1** | 61 | hell taco bell close midnight illeg mean pract... | 5 |
| **2** | 62 | actual intent visit place disgust next door ho... | 5 |
| **3** | 63 | yesterday around 3 30 pm drive west pinnacl re... | 5 |
| **4** | 64 | dr fitzmauric surgeri hand day 8 plu year ago ... | 5 |

### *Load previous tokenizer*

In [119]:
```
X = c6['text'].fillna('').values
y = pd.get_dummies(c6['stars'])

# with open('tokenizer.pickle', 'rb') as handle:
#     tokenizer = pickle.load(handle)

max_words

necc_cols = [1, 2, 3, 4, 5]
for col in necc_cols:
    if col not in y.columns:
        y[col] = 0

y = y[necc_cols]
y = y.values

X_baseline = tokenizer.texts_to_matrix(X)
X_lstm = tokenizer.texts_to_sequences(X)
X_lstm = pad_sequences(X_lstm, maxlen=400)
```

***Load and compile models***

In [120]:
```
# # Baseline
# baseline = load_model('./models/baseline.h5')

# baseline.compile(loss='categorical_crossentropy',
#                 optimizer=optimizer,
#                 metrics=['accuracy'])

# # LSTM
# lstm = load_model('./models/lstm.h5')

# lstm.compile(loss='categorical_crossentropy',
#                 optimizer=optimizer,
#                 metrics=['accuracy'])


# # One vs. all
# lstm_1 = load_model('./models/one_star.h5')

# lstm_1.compile(loss='binary_crossentropy',
#                     optimizer=optimizer,
#                     metrics=['accuracy'])

# lstm_2 = load_model('./models/two_star.h5')

# lstm_2.compile(loss='binary_crossentropy',
#                     optimizer=optimizer,
#                     metrics=['accuracy'])

# lstm_3 = load_model('./models/three_star.h5')

# lstm_3.compile(loss='binary_crossentropy',
#                     optimizer=optimizer,
#                     metrics=['accuracy'])

# lstm_4 = load_model('./models/four_star.h5')

# lstm_4.compile(loss='binary_crossentropy',
#                     optimizer=optimizer,
#                     metrics=['accuracy'])

# lstm_5 = load_model('./models/five_star.h5')

# lstm_5.compile(loss='binary_crossentropy',
#                     optimizer=optimizer,
#                     metrics=['accuracy'])
```

### *Evaluate Models*

In [121]:
```
# Baseline
print(baseline.evaluate(X_baseline, y))

# LSTM
print(lstm.evaluate(X_lstm, y))

# One vs. All
one_star_ps = lstm_1.predict(X_lstm)
two_star_ps = lstm_2.predict(X_lstm)
three_star_ps = lstm_3.predict(X_lstm)
four_star_ps = lstm_4.predict(X_lstm)
five_star_ps = lstm_5.predict(X_lstm)

data = [one_star_ps.flatten(), two_star_ps.flatten(), three_star_ps.flatten(),
four_star_ps.flatten(), five_star_ps.flatten()]
cols = [1, 2, 3, 4, 5]
ps = pd.DataFrame(data=data, index=cols).T

ps["ova_pred"] = ps.idxmax(axis=1)

print([MAE(ps["ova_pred"], pd.DataFrame(data=y, columns=cols).idxmax(axis=1)),
Accuracy(ps["ova_pred"], pd.DataFrame(data=y, columns=cols).idxmax(axis=1))])
```

```
500/500 [==============================] - 0s 76us/step
[0.24154025149345398, 0.4480000138282776, 0.22501754760742188]
500/500 [==============================] - 0s 642us/step
[0.214950764298439, 0.4399999976158142, 0.21064692735671997]
[2.188, 0.326]
```

#### *Attempt Ensemble*

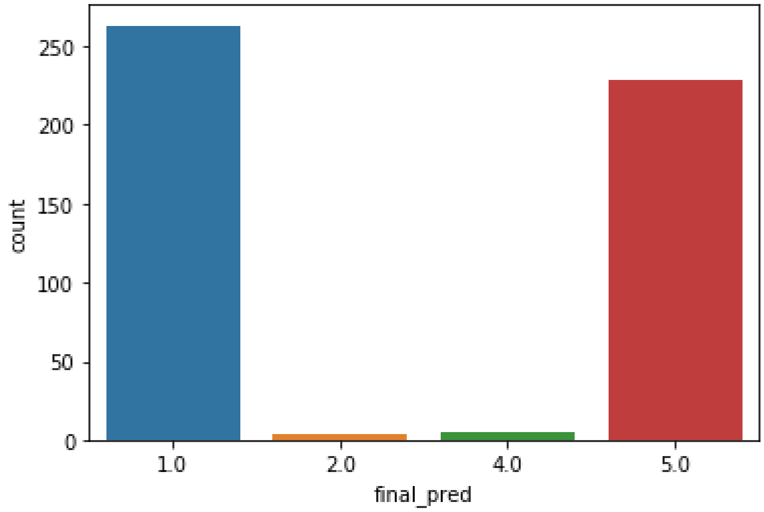
In [122]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# LSTM
lstm_preds = pd.DataFrame(lstm.predict(X_lstm), columns=cols)
lstm_preds['lstm_pred'] = lstm_preds.idxmax(axis=1)

# One vs. all
ova_preds = ps

all_preds = pd.DataFrame([baseline_preds['baseline_pred'], lstm_preds['lstm_pr
ed'], ova_preds['ova_pred']]).T
all_preds["final_pred"] = all_preds.mode(axis=1)[0]

print([MAE(all_preds["final_pred"], pd.DataFrame(data=y, columns=cols).idxmax(
axis=1)), Accuracy(all_preds["final_pred"], pd.DataFrame(data=y, columns=cols)
.idxmax(axis=1))])
```

```
[2.212, 0.438]
```

### *Misc.*

In [123]:
```
sns.countplot(all_preds["final_pred"])
```

Out[123]: `<matplotlib.axes._subplots.AxesSubplot at 0x25ba6d3eb88>`




## Challenge 3

In [124]:
```
c3 = pd.read_json("./yelp_challenge_3_with_answers.jsonl", lines = True)
print(c3.shape)
c3.head()
```

```
(534, 3)
```

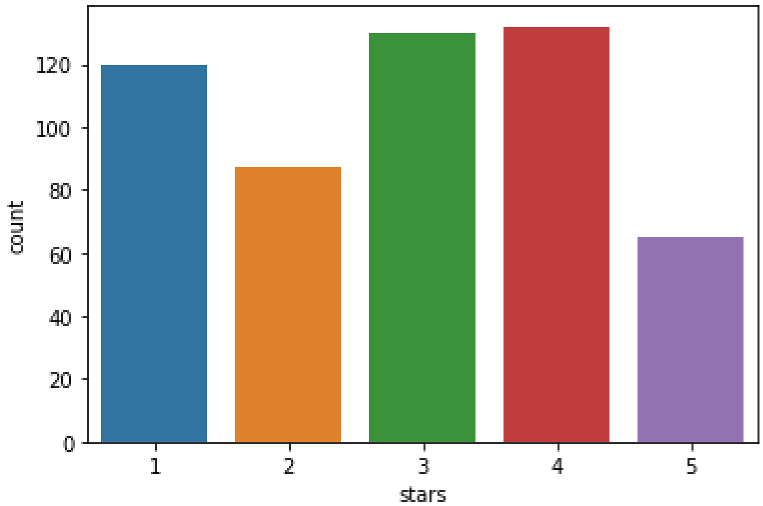
Out[124]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | 30 | We stopped here for lunch today and were pleas... | 4 |
| **1** | 31 | We went for a quick lunch here - it's all reas... | 3 |
| **2** | 32 | Very bad food, avoid it. We were a group of 4 ... | 2 |
| **3** | 33 | Bring a friend or two to help open the door. I... | 3 |
| **4** | 34 | Ukai serves some of the best sushi and sashimi... | 4 |

### *Quick EDA*

In [125]:
```
sns.countplot(c3['stars'])
```

Out[125]: <matplotlib.axes._subplots.AxesSubplot at 0x25ba6d13e48>




### *Pre-processing*

In [126]:
```
c3['text'] = c3['text'].apply(clean_text)
c3.head()
```

Out[126]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | 30 | stop lunch today pleasantli surpris great ambi... | 4 |
| **1** | 31 | went quick lunch reason well price good food n... | 3 |
| **2** | 32 | bad food avoid group 4 hungri came order batat... | 2 |
| **3** | 33 | bring friend two help open door think weigh 40... | 3 |
| **4** | 34 | ukai serv best sushi sashimi london bar nobu i... | 4 |

### *Load previous tokenizer*

In [127]:
```
X = c3['text'].fillna('').values
y = pd.get_dummies(c3['stars'])

# with open('tokenizer.pickle', 'rb') as handle:
#     tokenizer = pickle.load(handle)

max_words

necc_cols = [1, 2, 3, 4, 5]
for col in necc_cols:
    if col not in y.columns:
        y[col] = 0

y = y[necc_cols]
y = y.values

X_baseline = tokenizer.texts_to_matrix(X)
X_lstm = tokenizer.texts_to_sequences(X)
X_lstm = pad_sequences(X_lstm, maxlen=400)
```

***Load and compile models***

In [ ]:
```
# # Baseline
# baseline = load_model('./models/baseline.h5')

# baseline.compile(loss='categorical_crossentropy',
#               optimizer=optimizer,
#               metrics=['accuracy'])

# # LSTM
# lstm = load_model('./models/lstm.h5')

# lstm.compile(loss='categorical_crossentropy',
#               optimizer=optimizer,
#               metrics=['accuracy'])


# # One vs. all
# lstm_1 = load_model('./models/one_star.h5')

# lstm_1.compile(loss='binary_crossentropy',
#                  optimizer=optimizer,
#                  metrics=['accuracy'])

# lstm_2 = load_model('./models/two_star.h5')

# lstm_2.compile(loss='binary_crossentropy',
#                  optimizer=optimizer,
#                  metrics=['accuracy'])

# lstm_3 = load_model('./models/three_star.h5')

# lstm_3.compile(loss='binary_crossentropy',
#                  optimizer=optimizer,
#                  metrics=['accuracy'])

# lstm_4 = load_model('./models/four_star.h5')

# lstm_4.compile(loss='binary_crossentropy',
#                  optimizer=optimizer,
#                  metrics=['accuracy'])

# lstm_5 = load_model('./models/five_star.h5')

# lstm_5.compile(loss='binary_crossentropy',
#                  optimizer=optimizer,
#                  metrics=['accuracy'])
```

### *Evaluate Models*

In [128]:
```
# Baseline
print(baseline.evaluate(X_baseline, y))

# LSTM
print(lstm.evaluate(X_lstm, y))

# One vs. All
one_star_ps = lstm_1.predict(X_lstm)
two_star_ps = lstm_2.predict(X_lstm)
three_star_ps = lstm_3.predict(X_lstm)
four_star_ps = lstm_4.predict(X_lstm)
five_star_ps = lstm_5.predict(X_lstm)

data = [one_star_ps.flatten(), two_star_ps.flatten(), three_star_ps.flatten(),
four_star_ps.flatten(), five_star_ps.flatten()]
cols = [1, 2, 3, 4, 5]
ps = pd.DataFrame(data=data, index=cols).T

ps["ova_pred"] = ps.idxmax(axis=1)

print([MAE(ps["ova_pred"], pd.DataFrame(data=y, columns=cols).idxmax(axis=1)),
Accuracy(ps["ova_pred"], pd.DataFrame(data=y, columns=cols).idxmax(axis=1))])
```

```
534/534 [==============================] - 0s 77us/step
[0.24447482299715392, 0.4307115972042084, 0.2279004454612732]
534/534 [==============================] - 0s 524us/step
[0.19852803569384728, 0.35580524802207947, 0.19422420859336853]
[0.9250936329588015, 0.3445692883895131]
```

### *Attempt Ensemble*

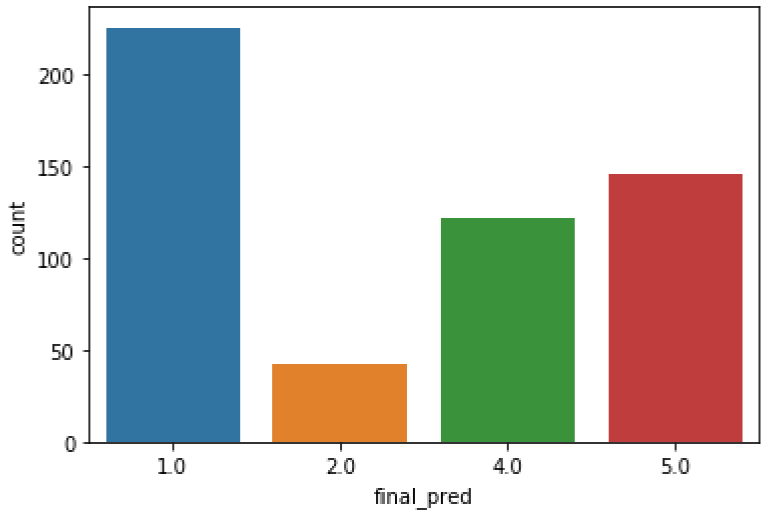
In [129]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# LSTM
lstm_preds = pd.DataFrame(lstm.predict(X_lstm), columns=cols)
lstm_preds['lstm_pred'] = lstm_preds.idxmax(axis=1)

# One vs. all
ova_preds = ps

all_preds = pd.DataFrame([baseline_preds['baseline_pred'], lstm_preds['lstm_pr
ed'], ova_preds['ova_pred']]).T
all_preds["final_pred"] = all_preds.mode(axis=1)[0]

print([MAE(all_preds["final_pred"], pd.DataFrame(data=y, columns=cols).idxmax(
axis=1)), Accuracy(all_preds["final_pred"], pd.DataFrame(data=y, columns=cols)
.idxmax(axis=1))])
```

```
[0.897003745318352, 0.3707865168539326]
```

***Misc.***

In [130]:
```
sns.countplot(all_preds["final_pred"])
```

Out[130]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x25ba6ecf188>
```




## Challenge 8

In [131]:
```
c8 = pd.read_json("./yelp_challenge_8_with_answers.jsonl", lines = True)
print(c8.shape)
c8.head()
```

```
(500, 3)
```

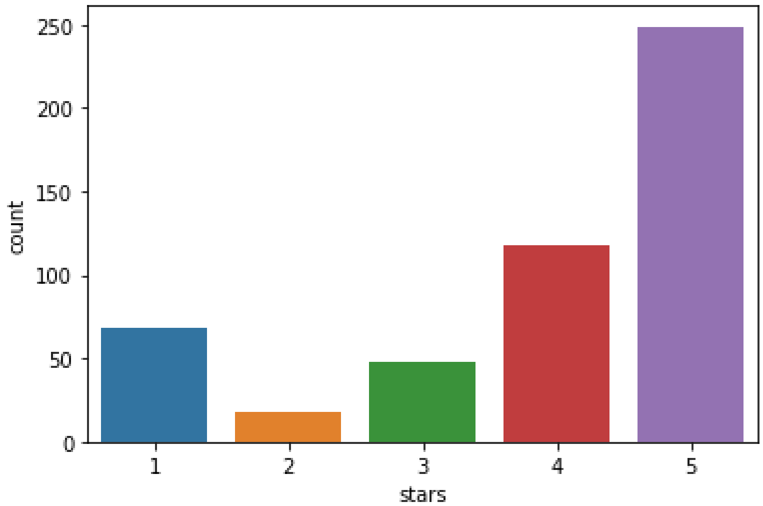
Out[131]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | qOOv-A-vo3kMT0yi4jIllg | Not bad for fast food. | 4 |
| **1** | uqxkO6B6w_sIDSAGr0k_0A | Une institution du café | 4 |
| **2** | 0o_gGSU0m_4QyNLWEHKgug | J ai vraiment aimé !!!! | 4 |
| **3** | BKAj-fKWW5G3yt3xAkbUCQ | They have good poutine. | 4 |
| **4** | fAhp8IwuGNT0ywKmsCs6VQ | Very old and dirty vans. | 1 |

***Quick EDA***

In [132]:
```
sns.countplot(c8['stars'])
```

Out[132]: <matplotlib.axes._subplots.AxesSubplot at 0x25ba6e8f548>




### *Pre-processing*

In [133]:
```
c8['text'] = c8['text'].apply(clean_text)
c8.head()
```

```
C:\Users\Tanner\Anaconda3\envs\yelp\lib\site-packages\bs4\__init__.py:398: Us
erWarning: "https://casetext.com/case/united-states-v-butterbaugh-2" looks li
ke a URL. Beautiful Soup is not an HTTP client. You should probably use an HT
TP client like requests to get the document behind the URL, and feed that doc
ument to Beautiful Soup.
  markup
```

Out[133]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | qOOv-A-vo3kMT0yi4jIllg | bad fast food | 4 |
| **1** | uqxkO6B6w_sIDSAGr0k_0A | une institut du caf | 4 |
| **2** | 0o_gGSU0m_4QyNLWEHKgug | j ai vraiment aim | 4 |
| **3** | BKAj-fKWW5G3yt3xAkbUCQ | good poutin | 4 |
| **4** | fAhp8IwuGNT0ywKmsCs6VQ | old dirti van | 1 |

### *Load previous tokenizer*

In [134]:
```
X = c8['text'].fillna('').values
y = pd.get_dummies(c8['stars'])

# with open('tokenizer.pickle', 'rb') as handle:
#     tokenizer = pickle.load(handle)

max_words

necc_cols = [1, 2, 3, 4, 5]
for col in necc_cols:
    if col not in y.columns:
        y[col] = 0

y = y[necc_cols]
y = y.values

X_baseline = tokenizer.texts_to_matrix(X)
X_lstm = tokenizer.texts_to_sequences(X)
X_lstm = pad_sequences(X_lstm, maxlen=400)
```

#### ***Load and compile models***

In [ ]:
```
# # Baseline
# baseline = load_model('./models/baseline.h5')

# baseline.compile(loss='categorical_crossentropy',
#                optimizer=optimizer,
#                metrics=['accuracy'])

# # LSTM
# lstm = load_model('./models/lstm.h5')

# lstm.compile(loss='categorical_crossentropy',
#                optimizer=optimizer,
#                metrics=['accuracy'])


# # One vs. all
# lstm_1 = load_model('./models/one_star.h5')

# lstm_1.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_2 = load_model('./models/two_star.h5')

# lstm_2.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_3 = load_model('./models/three_star.h5')

# lstm_3.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_4 = load_model('./models/four_star.h5')

# lstm_4.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])

# lstm_5 = load_model('./models/five_star.h5')

# lstm_5.compile(loss='binary_crossentropy',
#                    optimizer=optimizer,
#                    metrics=['accuracy'])
```

### ***Evaluate Models***

In [135]:
```
# Baseline
print(baseline.evaluate(X_baseline, y))

# LSTM
print(lstm.evaluate(X_lstm, y))

# One vs. All
one_star_ps = lstm_1.predict(X_lstm)
two_star_ps = lstm_2.predict(X_lstm)
three_star_ps = lstm_3.predict(X_lstm)
four_star_ps = lstm_4.predict(X_lstm)
five_star_ps = lstm_5.predict(X_lstm)

data = [one_star_ps.flatten(), two_star_ps.flatten(), three_star_ps.flatten(),
four_star_ps.flatten(), five_star_ps.flatten()]
cols = [1, 2, 3, 4, 5]
ps = pd.DataFrame(data=data, index=cols).T

ps["ova_pred"] = ps.idxmax(axis=1)

print([MAE(ps["ova_pred"], pd.DataFrame(data=y, columns=cols).idxmax(axis=1)),
Accuracy(ps["ova_pred"], pd.DataFrame(data=y, columns=cols).idxmax(axis=1))])
```

```
500/500 [==============================] - 0s 70us/step
[0.17787137413024903, 0.6079999804496765, 0.16142094135284424]
500/500 [==============================] - 0s 508us/step
[0.14866352832317353, 0.593999981880188, 0.14435969293117523]
[0.804, 0.538]
```

### *Attempt Ensemble*

In [136]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# LSTM
lstm_preds = pd.DataFrame(lstm.predict(X_lstm), columns=cols)
lstm_preds['lstm_pred'] = lstm_preds.idxmax(axis=1)

# One vs. all
ova_preds = ps

all_preds = pd.DataFrame([baseline_preds['baseline_pred'], lstm_preds['lstm_pr
ed'], ova_preds['ova_pred']]).T
all_preds["final_pred"] = all_preds.mode(axis=1)[0]

print([MAE(all_preds["final_pred"], pd.DataFrame(data=y, columns=cols).idxmax(
axis=1)), Accuracy(all_preds["final_pred"], pd.DataFrame(data=y, columns=cols)
.idxmax(axis=1))])
```

```
[0.684, 0.596]
```

***Misc.***

In [137]:
```
sns.countplot(all_preds["final_pred"])
```

Out[137]: <matplotlib.axes._subplots.AxesSubplot at 0x25ba6ee93c8>