# 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)) 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
  - One vs. All LSTM Approach
- Exploring Challenges
  - Challenge 5
  - Challenge 6

## Importing necessary libraries

In [ ]:
```
# 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
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
```

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

How large is the data?

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

## 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 [ ]:
```
yelp.isna().sum()
```

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

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 [ ]:
```
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)
```

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 [ ]:
```
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"])
    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()) # keeping
 all words, no stop word removal
#     new_text = ' '.join(ps.stem(word) for word in new_text.split() if word n
ot in STOPWORDS) # delete stopwords from text and stem
    return new_text

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

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

In [ ]:
```
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_1)
```

# 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 [49]:
```
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)
```

## Train/Test Split (Unbalanced and balanced)

In [50]:
```
yelp = pd.read_csv('cleaned_yelp_stemmed.csv')
yelp.head()
```

Out[50]:

| | Unnamed: 0 | review_id | text | stars | category |
|---|---|---|---|---|---|
| **0** | 0 | Q1sbwvVQXV2734tPgoKj4Q | total bill for thi horribl servic over 8g thes... | 1 | Negative |
| **1** | 1 | GJXCdrto3ASJOqKeVWPi6Q | i ador travi at the hard rock s new kelli card... | 5 | Positive |
| **2** | 2 | 2TzJjDVDEuAW6MR5Vuc1ug | i have to say that thi offic realli ha it toge... | 5 | Positive |
| **3** | 3 | yi0R0Ugj_xUx_Nek0-_Qig | went in for a lunch steak sandwich wa delici a... | 5 | Positive |
| **4** | 4 | 11a8sVPMUFtaC7_ABRkmtw | today wa my second out of three session i had ... | 1 | Negative |

In [51]:
```
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)
```

## Baseline Sequential Model

In [52]:
```
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)
```

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

In [ ]:
```
# 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)
```

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 [53]:
```
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='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

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

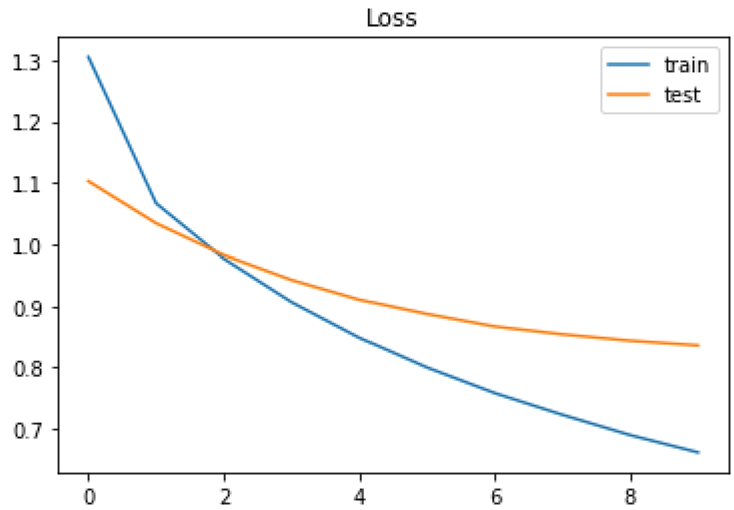
```
Train on 298804 samples, validate on 74702 samples
Epoch 1/10
298804/298804 [==============================] - 22s 73us/step - loss: 1.3055
- accuracy: 0.6878 - val_loss: 1.1030 - val_accuracy: 0.7437
Epoch 2/10
298804/298804 [==============================] - 12s 40us/step - loss: 1.0671
- accuracy: 0.7486 - val_loss: 1.0346 - val_accuracy: 0.7491
Epoch 3/10
298804/298804 [==============================] - 12s 39us/step - loss: 0.9772
- accuracy: 0.7633 - val_loss: 0.9830 - val_accuracy: 0.7512
Epoch 4/10
298804/298804 [==============================] - 12s 39us/step - loss: 0.9067
- accuracy: 0.7749 - val_loss: 0.9426 - val_accuracy: 0.7507
Epoch 5/10
298804/298804 [==============================] - 12s 39us/step - loss: 0.8487
- accuracy: 0.7843 - val_loss: 0.9106 - val_accuracy: 0.7525
Epoch 6/10
298804/298804 [==============================] - 11s 38us/step - loss: 0.8001
- accuracy: 0.7921 - val_loss: 0.8874 - val_accuracy: 0.7518
Epoch 7/10
298804/298804 [==============================] - 12s 39us/step - loss: 0.7584
- accuracy: 0.8002 - val_loss: 0.8670 - val_accuracy: 0.7508
Epoch 8/10
298804/298804 [==============================] - 12s 39us/step - loss: 0.7230
- accuracy: 0.8077 - val_loss: 0.8542 - val_accuracy: 0.7490
Epoch 9/10
298804/298804 [==============================] - 12s 39us/step - loss: 0.6899
- accuracy: 0.8165 - val_loss: 0.8438 - val_accuracy: 0.7509
Epoch 10/10
298804/298804 [==============================] - 12s 39us/step - loss: 0.6618
- accuracy: 0.8232 - val_loss: 0.8362 - val_accuracy: 0.7511
```

In [54]:

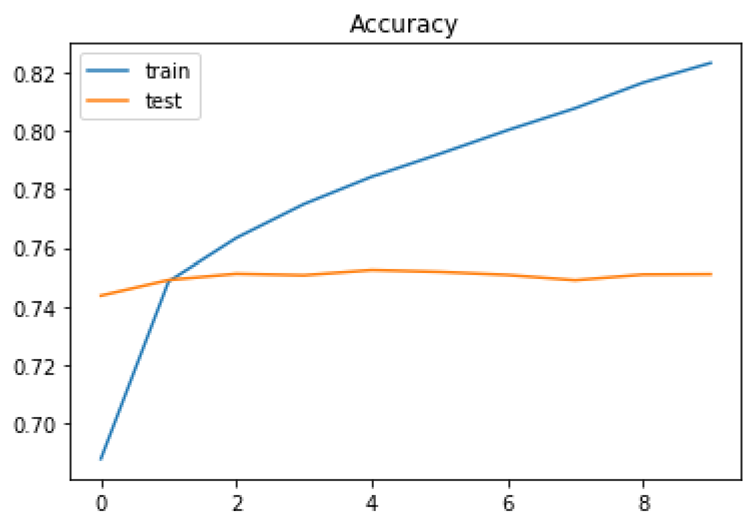
```
score = baseline.evaluate(X_test, y_test,
                       batch_size=batch_size, verbose=1)
print('Test accuracy:', score[1])
```

```
160075/160075 [==============================] - 13s 80us/step
Test accuracy: 0.7536966800689697
```

In [55]:
```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```




In [56]:
```
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()
```




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 [33]:
```
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)
```

### LSTM #1

In [34]:
```
batch_size = 512
epochs = 10

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='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

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 1/10
298804/298804 [==============================] - 87s 292us/step - loss: 0.789
7 - accuracy: 0.7041 - val_loss: 0.6659 - val_accuracy: 0.7420
Epoch 2/10
298804/298804 [==============================] - 87s 292us/step - loss: 0.649
7 - accuracy: 0.7528 - val_loss: 0.6169 - val_accuracy: 0.7640
Epoch 3/10
298804/298804 [==============================] - 85s 285us/step - loss: 0.609
1 - accuracy: 0.7650 - val_loss: 0.6040 - val_accuracy: 0.7672
Epoch 4/10
298804/298804 [==============================] - 85s 285us/step - loss: 0.587
9 - accuracy: 0.7719 - val_loss: 0.5929 - val_accuracy: 0.7698
Epoch 5/10
298804/298804 [==============================] - 85s 286us/step - loss: 0.572
4 - accuracy: 0.7776 - val_loss: 0.5925 - val_accuracy: 0.7728
Epoch 6/10
298804/298804 [==============================] - 85s 284us/step - loss: 0.560
3 - accuracy: 0.7828 - val_loss: 0.5810 - val_accuracy: 0.7765
Epoch 7/10
298804/298804 [==============================] - 86s 287us/step - loss: 0.549
0 - accuracy: 0.7872 - val_loss: 0.5755 - val_accuracy: 0.7785
Epoch 8/10
298804/298804 [==============================] - 85s 284us/step - loss: 0.539
1 - accuracy: 0.7906 - val_loss: 0.5791 - val_accuracy: 0.7777
Epoch 9/10
298804/298804 [==============================] - 85s 284us/step - loss: 0.531
1 - accuracy: 0.7942 - val_loss: 0.5778 - val_accuracy: 0.7806
Epoch 10/10
298804/298804 [==============================] - 85s 284us/step - loss: 0.522
9 - accuracy: 0.7981 - val_loss: 0.5802 - val_accuracy: 0.7784
```

### LSTM #1: Evaluation

In [35]:

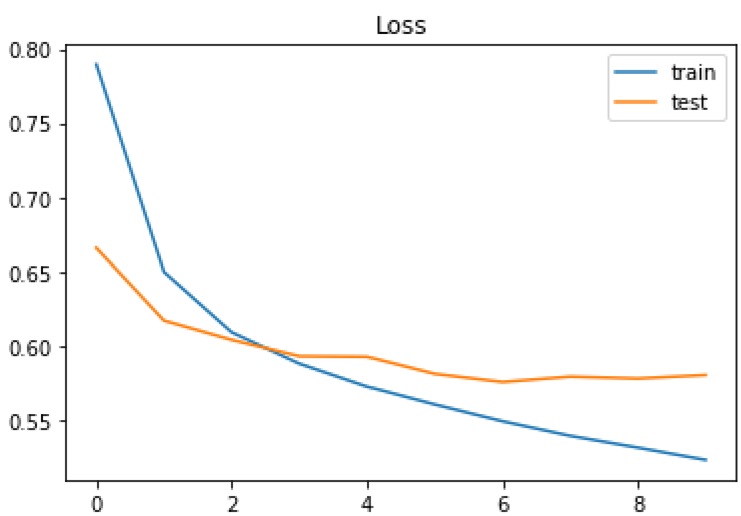
```
score = lstm.evaluate(X_test, y_test,
                       batch_size=batch_size, verbose=1)
print('Test accuracy:', score[1])
```

```
160075/160075 [==============================] - 11s 69us/step
Test accuracy: 0.7798094749450684
```

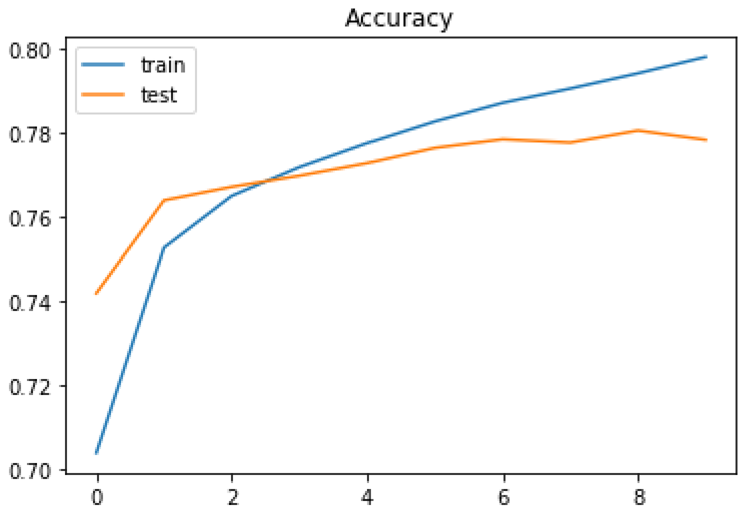
In [36]:
```
lstm.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_2 (Embedding)      (None, 400, 128)          384000    
_________________________________________________________________
spatial_dropout1d_2 (Spatial (None, 400, 128)          0         
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 396, 64)           41024     
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 99, 64)            0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               98816     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dense_6 (Dense)              (None, 5)                 645       
=================================================================
Total params: 524,997
Trainable params: 524,741
Non-trainable params: 256
_________________________________________________________________
```

In [37]:
```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```



In [38]:
```
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.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 [39]:
```
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 [40]:
```
stars = np.arange(1, 6)
models = {}
histories = {}
batch_size = 1024

for star in stars:
    if star in [1]:
        epochs = 2
    elif star in [2, 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='binary_crossentropy',
                 optimizer=optimizer,
                 metrics=['accuracy'])

    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 1/3
298804/298804 [==============================] - 67s 226us/step - loss: 0.250
9 - accuracy: 0.9039 - val_loss: 0.5960 - val_accuracy: 0.7587
Epoch 2/3
298804/298804 [==============================] - 66s 222us/step - loss: 0.178
2 - accuracy: 0.9334 - val_loss: 0.2453 - val_accuracy: 0.8876
Epoch 3/3
298804/298804 [==============================] - 66s 220us/step - loss: 0.156
4 - accuracy: 0.9423 - val_loss: 0.3240 - val_accuracy: 0.8869
2
Train on 298804 samples, validate on 74702 samples
Epoch 1/3
298804/298804 [==============================] - 67s 224us/step - loss: 0.256
8 - accuracy: 0.9182 - val_loss: 0.2734 - val_accuracy: 0.9323
Epoch 2/3
298804/298804 [==============================] - 66s 221us/step - loss: 0.182
3 - accuracy: 0.9353 - val_loss: 0.2096 - val_accuracy: 0.9325
Epoch 3/3
298804/298804 [==============================] - 66s 220us/step - loss: 0.163
2 - accuracy: 0.9402 - val_loss: 0.2037 - val_accuracy: 0.9342
3
Train on 298804 samples, validate on 74702 samples
Epoch 1/3
298804/298804 [==============================] - 67s 224us/step - loss: 0.251
0 - accuracy: 0.9214 - val_loss: 0.2580 - val_accuracy: 0.9363
Epoch 2/3
298804/298804 [==============================] - 66s 221us/step - loss: 0.176
4 - accuracy: 0.9391 - val_loss: 0.2564 - val_accuracy: 0.9363
Epoch 3/3
298804/298804 [==============================] - 66s 222us/step - loss: 0.157
9 - accuracy: 0.9444 - val_loss: 0.1955 - val_accuracy: 0.9390
4
Train on 298804 samples, validate on 74702 samples
Epoch 1/3
298804/298804 [==============================] - 67s 224us/step - loss: 0.360
6 - accuracy: 0.8547 - val_loss: 0.4197 - val_accuracy: 0.8639
Epoch 2/3
298804/298804 [==============================] - 66s 223us/step - loss: 0.304
1 - accuracy: 0.8743 - val_loss: 0.3464 - val_accuracy: 0.8640
Epoch 3/3
298804/298804 [==============================] - 66s 221us/step - loss: 0.283
0 - accuracy: 0.8831 - val_loss: 0.3158 - val_accuracy: 0.8696
5
Train on 298804 samples, validate on 74702 samples
Epoch 1/3
298804/298804 [==============================] - 67s 224us/step - loss: 0.347
0 - accuracy: 0.8598 - val_loss: 0.5140 - val_accuracy: 0.7406
Epoch 2/3
298804/298804 [==============================] - 67s 223us/step - loss: 0.287
5 - accuracy: 0.8851 - val_loss: 0.3091 - val_accuracy: 0.8734
Epoch 3/3
298804/298804 [==============================] - 66s 221us/step - loss: 0.260
3 - accuracy: 0.8973 - val_loss: 0.2885 - val_accuracy: 0.8816
```

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

#### *Testing*

In [41]:
```
%%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
0.38773699828205527
0.7536592222395752
Wall time: 5min 50s
```

### 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 [42]:
```
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 [43]:
```
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 [44]:
```
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))])
```

```
[0.33215055442761204, 0.7702264563485866]
```

# Challenges

## Challenge 5

In [45]:
```
c5 = pd.read_json("./yelp_challenge_5_with_answers.jsonl", lines = True)
print(c5.shape)
c5.head()
```

```
(500, 3)
```

Out[45]:

| | 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 [46]:
```
sns.countplot(c5['stars'])
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x238037a3508>
```


### *Pre-processing*

In [47]:
```
c5['text'] = c5['text'].apply(clean_text)
c5.head()
```

Out[47]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | 50 | i went to thi campu for 1 semest i wa in busi ... | 2 |
| **1** | 51 | i have rate it a two star base on it compariso... | 2 |
| **2** | 52 | just like most of the review we order and paid... | 2 |
| **3** | 53 | i onli go here if it is an emerg i hate it tha... | 2 |
| **4** | 54 | rude staff i got 60 feeder fish and about 15 w... | 2 |

#### ***Load previous tokenizer***

In [58]:
```
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 [59]:
```
# 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 86us/step
[2.0396031694412233, 0.2720000147819519]
500/500 [==============================] - 1s 1ms/step
[1.3660213603973388, 0.40400001406669617]
[0.988, 0.094]
```

### *Attempt Ensemble*

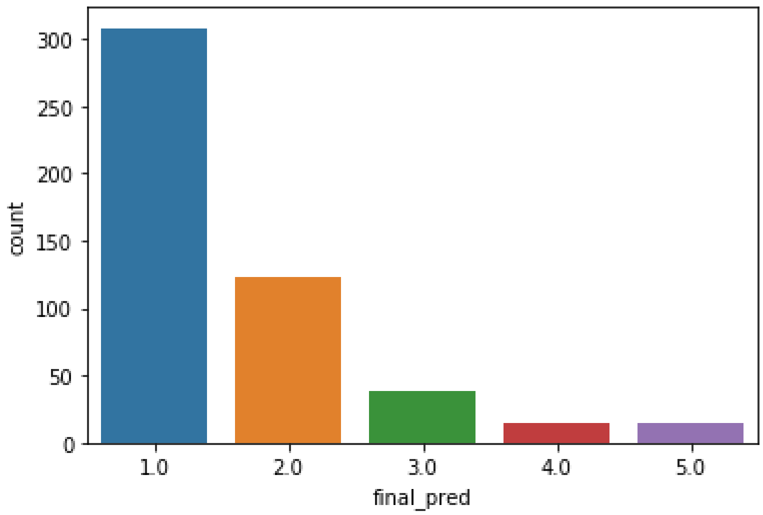
In [60]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# 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))])
```

```
[0.844, 0.246]
```

### *Misc.*

In [61]:
```
sns.countplot(all_preds["final_pred"])
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x239868a9148>




## Challenge 6

In [62]:
```
c6 = pd.read_json("./yelp_challenge_6_with_answers.jsonl", lines = True)
print(c6.shape)
c6.head()
```

(500, 3)

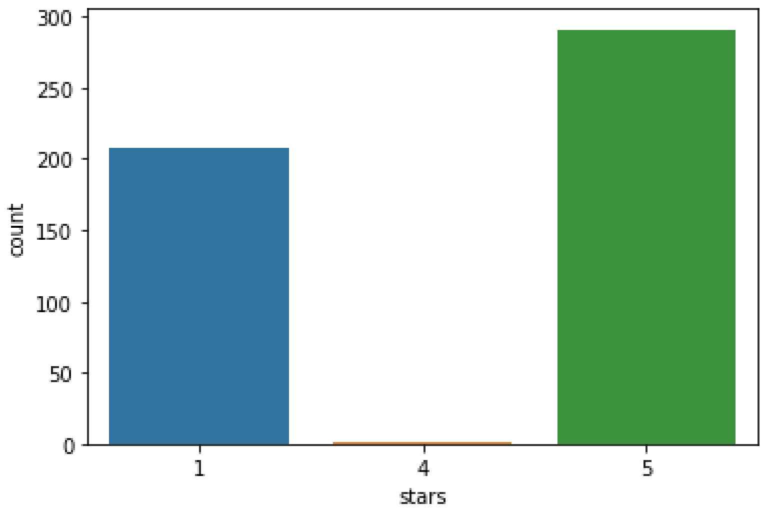
Out[62]:

| | 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 [63]:
```
sns.countplot(c6['stars'])
```

Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x237b1342a48>




### *Pre-processing*

In [64]:
```
c6['text'] = c6['text'].apply(clean_text)
c6.head()
```

Out[64]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | 60 | amaz for tree 20 for a 5 gallon i will never g... | 5 |
| **1** | 61 | how the hell can taco bell be close befor midn... | 5 |
| **2** | 62 | i actual had no intent of visit thi place at a... | 5 |
| **3** | 63 | yesterday around 3 30 pm i wa drive west on pi... | 5 |
| **4** | 64 | dr fitzmauric did surgeri on both hand on the ... | 5 |

### *Load previous tokenizer*

In [65]:
```
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 [ ]:
```
# 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 [66]:
```
# 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 100us/step
[2.3629399967193603, 0.4339999854564667]
500/500 [==============================] - 0s 530us/step
[2.310380277633667, 0.44200000166893005]
[2.196, 0.446]
```

#### *Attempt Ensemble*

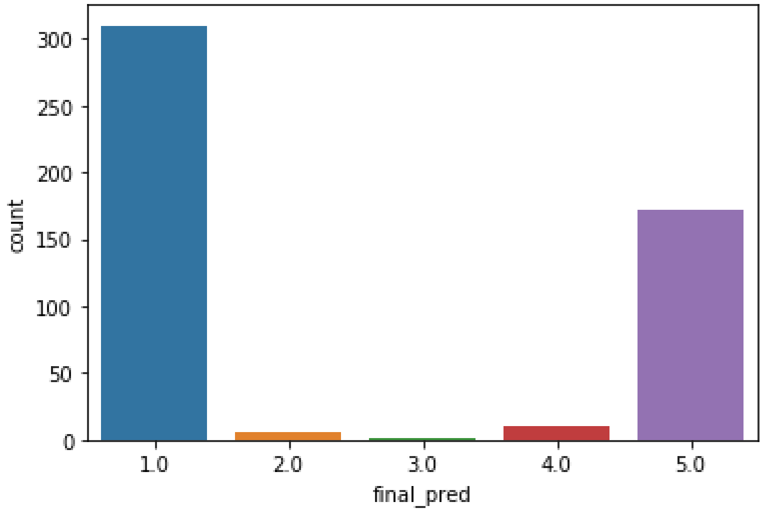
In [67]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# 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))])
```

```
[2.07, 0.464]
```

### *Misc.*

In [68]:
```
sns.countplot(all_preds["final_pred"])
```

Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x237b13b30c8>




## Challenge 3

In [69]:
```
c3 = pd.read_json("./yelp_challenge_3_with_answers.jsonl", lines = True)
print(c3.shape)
c3.head()
```

```
(534, 3)
```

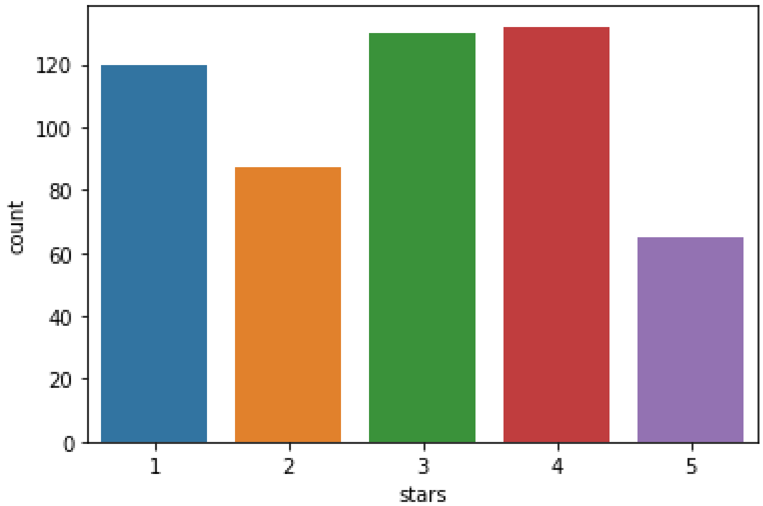
Out[69]:

| | 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 [70]:
```python
sns.countplot(c3['stars'])
```

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x237b138d948>




### *Pre-processing*

In [71]:
```python
c3['text'] = c3['text'].apply(clean_text)
c3.head()
```

Out[71]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | 30 | we stop here for lunch today and were pleasant... | 4 |
| **1** | 31 | we went for a quick lunch here it s all reason... | 3 |
| **2** | 32 | veri bad food avoid it we were a group of 4 an... | 2 |
| **3** | 33 | bring a friend or two to help open the door i ... | 3 |
| **4** | 34 | ukai serv some of the best sushi and sashimi i... | 4 |

### *Load previous tokenizer*

In [72]:
```
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 [73]:
```
# 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 92us/step
[1.1802164695682598, 0.567415714263916]
534/534 [==============================] - 0s 562us/step
[0.91535808955239, 0.6086142063140869]
[0.5823970037453183, 0.548689138576779]
```

### *Attempt Ensemble*

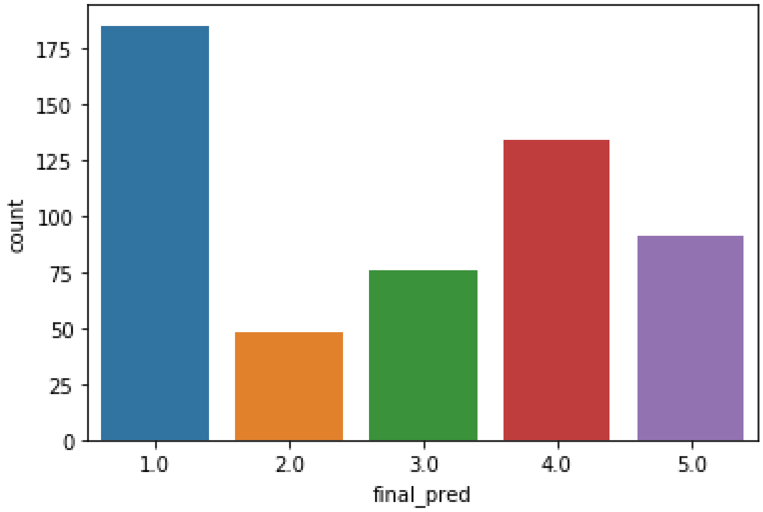
In [74]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# 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))])
```

```
[0.5074906367041199, 0.599250936329588]
```

### *Misc.*

In [75]:
```
sns.countplot(all_preds["final_pred"])
```

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x237b14dde48>




## Challenge 8

In [76]:
```
c8 = pd.read_json("./yelp_challenge_8_with_answers.jsonl", lines = True)
print(c8.shape)
c8.head()
```

(500, 3)

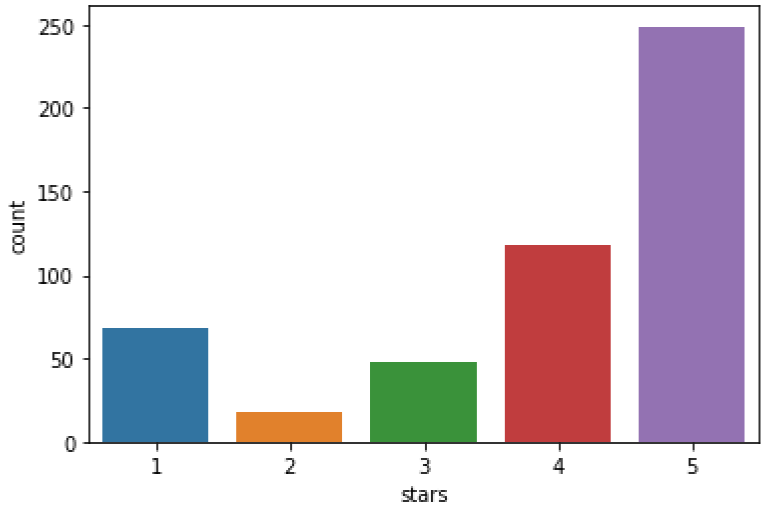
Out[76]:

| | 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 [77]:
```
sns.countplot(c8['stars'])
```

Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x237b13e2908>




### *Pre-processing*

In [78]:
```
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[78]:

| | review_id | text | stars |
|---|---|---|---|
| **0** | qOOv-A-vo3kMT0yi4jIllg | not bad for fast food | 4 |
| **1** | uqxkO6B6w_sIDSAGr0k_0A | une institut du caf | 4 |
| **2** | 0o_gGSU0m_4QyNLWEHKgug | j ai vraiment aim | 4 |
| **3** | BKAj-fKWW5G3yt3xAkbUCQ | they have good poutin | 4 |
| **4** | fAhp8IwuGNT0ywKmsCs6VQ | veri old and dirti van | 1 |

### *Load previous tokenizer*

In [79]:
```
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 [80]:
```
# 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 84us/step
[1.0328141555786132, 0.6439999938011169]
500/500 [==============================] - 0s 534us/step
[0.928207818031311, 0.6359999775886536]
[0.848, 0.574]
```

***Attempt Ensemble***

In [81]:
```
# 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))])
```

```
[0.624, 0.626]
```

### *Misc.*

In [82]:
```
sns.countplot(all_preds["final_pred"])
```

Out[82]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x237c49a9848>
```