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

Importing necessary libraries

In [2]:
```
# 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
```

```
Using TensorFlow backend.
```

In [3]:
```
yelp = pd.read_json("./yelp_review_training_dataset.jsonl", lines = True)
yelp.head()
```

Out[3]:

	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 [4]:
```
yelp.shape
```

Out[4]:
```
(533581, 3)
```

EDA - Stars

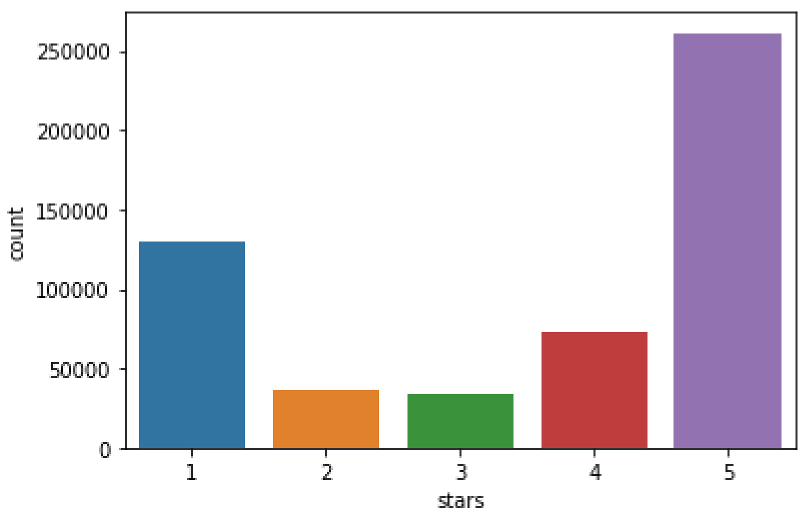
Not too much to go off of, but let's get a general understanding of our data. How many nulls do we have?

In [5]:
```
yelp.isna().sum()
```

Out[5]:
```
review_id    0
text         0
stars        0
dtype: int64
```

In [6]:
```
sns.countplot(yelp['stars'])
```

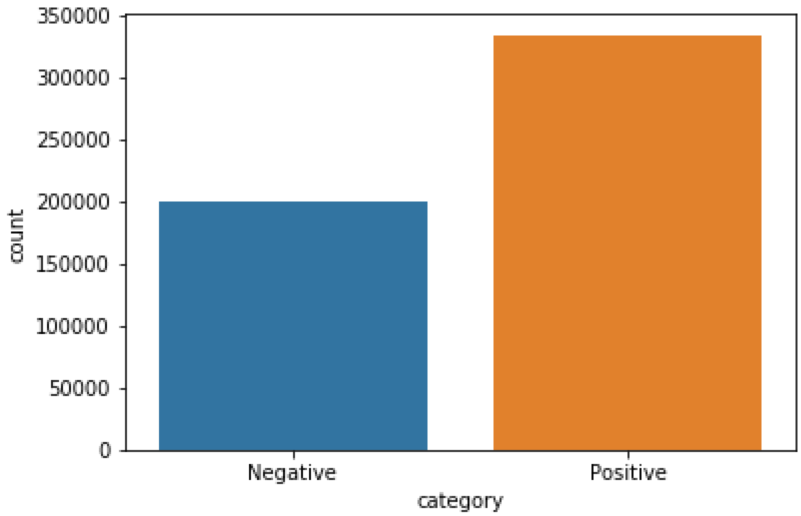
Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2277d9e9648>
```



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 [7]:
```
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 [8]:
```
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)
```

```
{'an', 'themselves', 'these', 'whom', 'yourselves', 'hers', 'o', "you'll", "h
adn't", 'how', 've', "shouldn't", 'those', 'of', "don't", 'myself', 'or', 'th
eirs', 'where', 'any', 'again', 'some', 'll', 'a', 'from', 'your', 'if', 'm
a', 'above', 'there', 'i', 'are', "that'll", 'with', 't', 'through', 'y', 'be
ing', 'no', 'their', "wouldn't", 'most', 'what', 'that', "hasn't", 'then', 't
hey', 'don', 'and', 'who', 'under', 'when', 'can', "didn't", 'over', 'such',
'only', "haven't", "aren't", 'for', "it's", 'few', 'mightn', 'between', 'sh
e', 'should', 'aren', 'was', 'very', 'has', 'below', 'd', 'to', 'didn', "yo
u've", "needn't", 'her', 'its', 'each', 'himself', 'him', 'he', 'once', 'tha
n', "couldn't", 'mustn', 'off', "mightn't", 'be', 'during', 'so', 'hadn', 'no
r', 'which', "she's", 'this', 'itself', 'been', 'did', 'other', 'shan', 'doe
s', 'we', 'all', 'not', 'into', 'weren', 'me', 're', 'having', 'doing', 'hers
elf', 'further', 'up', 'on', 'in', 'ain', 'now', 'by', "shan't", 'yours', 'we
re', 'why', 'is', "won't", "should've", 'hasn', 'needn', 'own', "wasn't", 'd
o', 'isn', 'had', 'am', 'them', "isn't", "you'd", 'couldn', 'against', 'bot
h', 's', 'yourself', 'more', 'too', 'wouldn', 'ours', 'our', "doesn't", 'was
n', 'it', 'because', 'haven', 'as', 'about', 'ourselves', 'before', 'here',
'just', 'doesn', 'down', 'the', 'at', 'same', "weren't", 'but', 'have', "yo
u're", 'my', 'won', 'm', 'while', 'you', 'shouldn', 'until', 'out', "must
n't", 'after', 'his', 'will'}
{'an', 'themselves', 'these', 'whom', 'yourselves', 'hers', 'o', "you'll", "h
adn't", 'how', 've', 'those', 'of', 'myself', 'or', 'theirs', 'where', 'any',
'again', 'some', 'll', 'a', 'from', 'your', 'if', 'ma', 'there', 'i', 'are',
"that'll", 'with', 't', 'through', 'y', 'being', 'their', 'most', 'what', 'th
at', 'then', 'they', 'don', 'and', 'who', 'when', 'can', 'such', 'only', "are
n't", 'for', "it's", 'between', 'she', 'should', 'was', 'has', 'd', 'to', "yo
u've", "needn't", 'her', 'its', 'each', 'himself', 'him', 'he', 'once', "coul
dn't", 'mustn', 'off', "mightn't", 'be', 'during', 'so', 'hadn', 'which', "sh
e's", 'this', 'itself', 'been', 'did', 'other', 'shan', 'does', 'we', 'all',
'into', 'weren', 'me', 're', 'having', 'doing', 'herself', 'further', 'up',
'on', 'in', 'now', 'by', 'yours', 'were', 'why', 'is', "should've", 'hasn',
'needn', 'own', 'do', 'isn', 'had', 'am', 'them', "you'd", 'couldn', 'agains
t', 'both', 's', 'yourself', 'more', 'ours', 'our', 'it', 'because', 'haven',
'as', 'about', 'ourselves', 'before', 'here', 'just', 'down', 'the', 'at', 's
ame', 'but', 'have', "you're", 'my', 'm', 'while', 'you', 'until', 'out', 'af
ter', 'his', 'will'}
```

In [ ]:

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

In [9]:
```
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[9]:
```
'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 [10]:
```
from keras.losses import mean_absolute_error, binary_crossentropy, categorical
_crossentropy

def my_custom_loss_ova(y_true, y_pred):
    mse = mean_squared_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_squared_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 [11]:
```
yelp = pd.read_csv('cleaned_yelp_stemmed.csv')
yelp.head()
```

Out[11]:

	Unnamed: 0	review_id	text	stars	category
0	0	Q1sbwvVQXV2734tPgoKj4Q	total bill horribl servic over 8g crook actual...	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 [12]:
```
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 [13]:
```
%%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: 49.9 s
```

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)
```

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)**

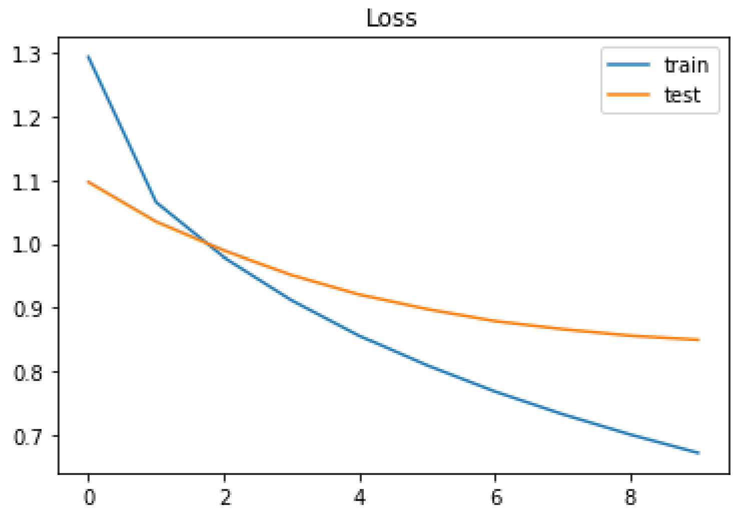
In [14]:
```
batch_size = 512
epochs = 10

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', '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 [==============================] - 25s 84us/step - loss: 1.2934
- accuracy: 0.6901 - mean_absolute_error: 0.1606 - val_loss: 1.0971 - val_acc
uracy: 0.7417 - val_mean_absolute_error: 0.1372
Epoch 2/10
298804/298804 [==============================] - 12s 39us/step - loss: 1.0654
- accuracy: 0.7456 - mean_absolute_error: 0.1342 - val_loss: 1.0349 - val_acc
uracy: 0.7470 - val_mean_absolute_error: 0.1339
Epoch 3/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.9794
- accuracy: 0.7607 - mean_absolute_error: 0.1292 - val_loss: 0.9898 - val_acc
uracy: 0.7479 - val_mean_absolute_error: 0.1320
Epoch 4/10
298804/298804 [==============================] - 11s 36us/step - loss: 0.9122
- accuracy: 0.7717 - mean_absolute_error: 0.1259 - val_loss: 0.9516 - val_acc
uracy: 0.7493 - val_mean_absolute_error: 0.1317
Epoch 5/10
298804/298804 [==============================] - 11s 36us/step - loss: 0.8563
- accuracy: 0.7805 - mean_absolute_error: 0.1230 - val_loss: 0.9210 - val_acc
uracy: 0.7486 - val_mean_absolute_error: 0.1321
Epoch 6/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.8099
- accuracy: 0.7888 - mean_absolute_error: 0.1203 - val_loss: 0.8981 - val_acc
uracy: 0.7484 - val_mean_absolute_error: 0.1316
Epoch 7/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.7688
- accuracy: 0.7974 - mean_absolute_error: 0.1175 - val_loss: 0.8793 - val_acc
uracy: 0.7484 - val_mean_absolute_error: 0.1305
Epoch 8/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.7331
- accuracy: 0.8050 - mean_absolute_error: 0.1147 - val_loss: 0.8667 - val_acc
uracy: 0.7491 - val_mean_absolute_error: 0.1281
Epoch 9/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.7012
- accuracy: 0.8129 - mean_absolute_error: 0.1118 - val_loss: 0.8564 - val_acc
uracy: 0.7481 - val_mean_absolute_error: 0.1289
Epoch 10/10
298804/298804 [==============================] - 11s 37us/step - loss: 0.6725
- accuracy: 0.8209 - mean_absolute_error: 0.1088 - val_loss: 0.8499 - val_acc
uracy: 0.7483 - val_mean_absolute_error: 0.1272
```

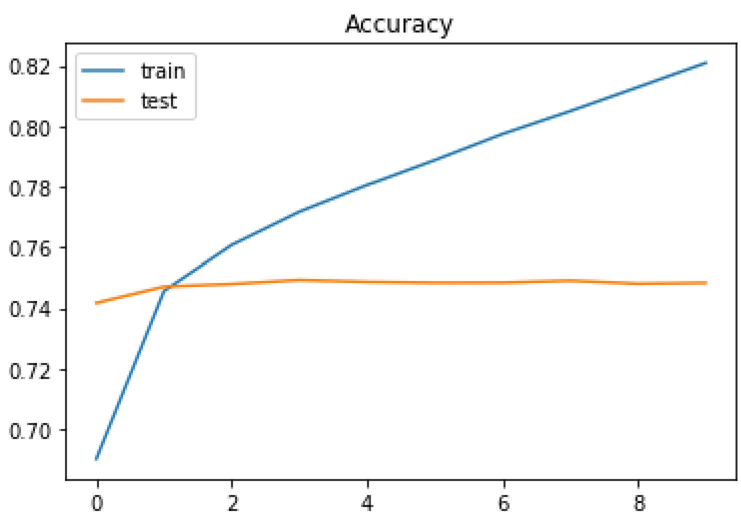
In [15]:
```
score = baseline.evaluate(X_test, y_test,
                          batch_size=batch_size, verbose=1)
print('Test accuracy:', score[1])
```

```
160075/160075 [==============================] - 11s 69us/step
Test accuracy: 0.7501671314239502
```

In [16]:
```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```



In [17]:
```
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()
```



In [18]:
```
# 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[18]:

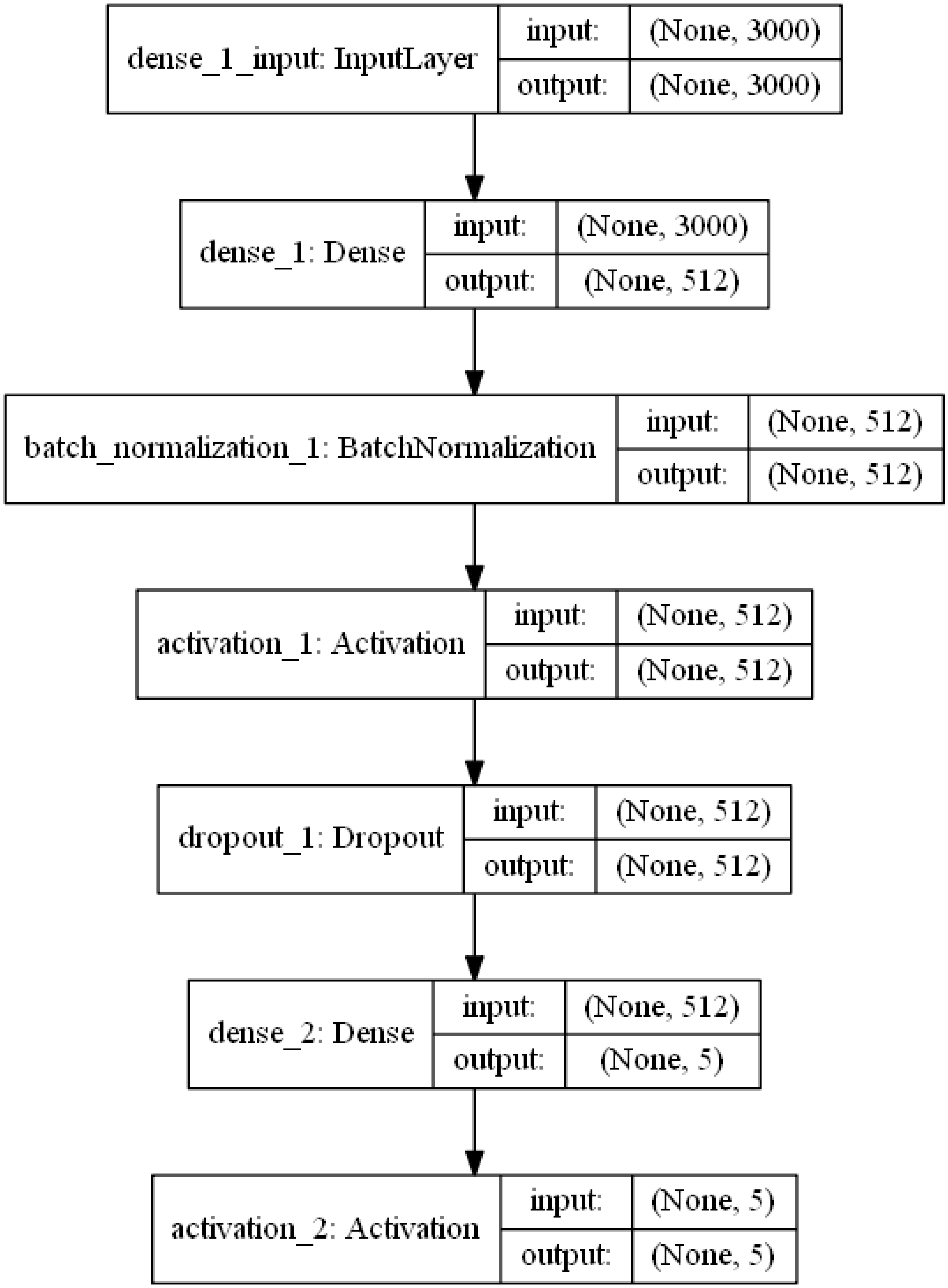
	1	2	3	4	5
1	35006	5213	1592	704	1295
2	1622	2482	1361	422	192
3	541	1492	2840	1682	509
4	316	678	2647	7492	4162
5	1402	878	1823	11461	72263

In [19]:
```
print(classification_report(y_pred_true, y_test_true))
```

```
              precision    recall  f1-score   support

           1       0.90      0.80      0.85     43810
           2       0.23      0.41      0.30      6079
           3       0.28      0.40      0.33      7064
           4       0.34      0.49      0.40     15295
           5       0.92      0.82      0.87     87827

    accuracy                           0.75    160075
   macro avg       0.53      0.58      0.55    160075
weighted avg       0.81      0.75      0.77    160075
```

In [20]:
```
plot_model(baseline, to_file='baseline.png', show_shapes=True)
```

Out[20]:



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 [21]:
```
%%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: 23.3 s
```

LSTM #1

In [25]:
```
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='categorical_crossentropy',
              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 274us/step - loss: 0.781
0 - accuracy: 0.7121 - mean_absolute_error: 0.1992 - val_loss: 0.6701 - val_a
ccuracy: 0.7441 - val_mean_absolute_error: 0.1467
Epoch 2/5
298804/298804 [==============================] - 81s 271us/step - loss: 0.652
9 - accuracy: 0.7509 - mean_absolute_error: 0.1445 - val_loss: 0.6295 - val_a
ccuracy: 0.7587 - val_mean_absolute_error: 0.1432
Epoch 3/5
298804/298804 [==============================] - 82s 276us/step - loss: 0.620
2 - accuracy: 0.7611 - mean_absolute_error: 0.1315 - val_loss: 0.6195 - val_a
ccuracy: 0.7611 - val_mean_absolute_error: 0.1302
Epoch 4/5
298804/298804 [==============================] - 84s 281us/step - loss: 0.599
2 - accuracy: 0.7682 - mean_absolute_error: 0.1270 - val_loss: 0.6135 - val_a
ccuracy: 0.7646 - val_mean_absolute_error: 0.1254
Epoch 5/5
298804/298804 [==============================] - 82s 274us/step - loss: 0.582
6 - accuracy: 0.7747 - mean_absolute_error: 0.1250 - val_loss: 0.6111 - val_a
ccuracy: 0.7659 - val_mean_absolute_error: 0.1323
```

LSTM #1: Evaluation

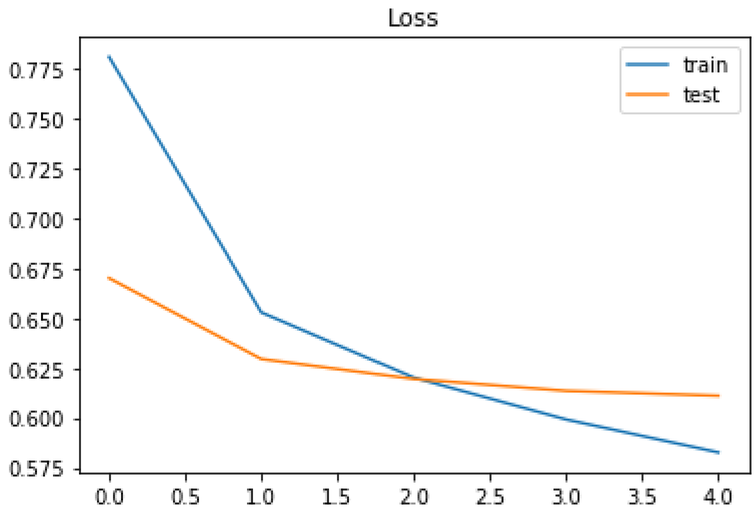
In [26]:
```
score = lstm.evaluate(X_test, y_test,
                       batch_size=batch_size, verbose=1)
print('Test accuracy:', score[1])
```

```
160075/160075 [==============================] - 11s 67us/step
Test accuracy: 0.7659659385681152
```

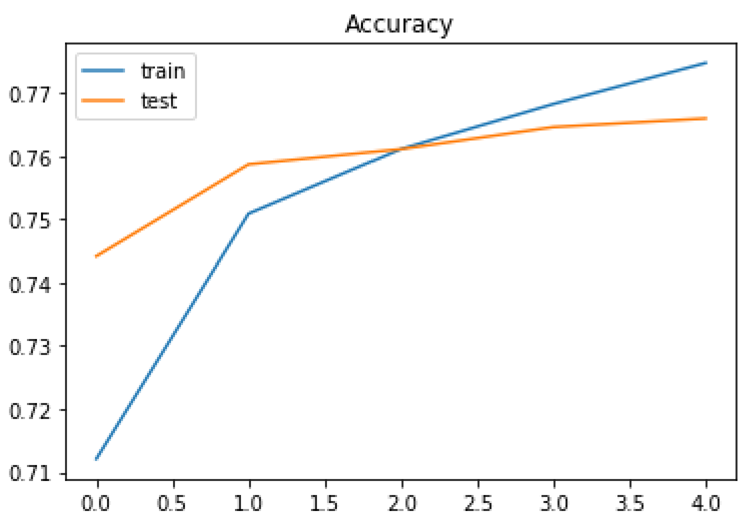
In [27]:
```
lstm.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 400, 128)          384000    
_________________________________________________________________
spatial_dropout1d_3 (Spatial (None, 400, 128)          0         
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 396, 64)           41024     
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 99, 64)            0         
_________________________________________________________________
lstm_3 (LSTM)                (None, 128)               98816     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 645       
=================================================================
Total params: 524,997
Trainable params: 524,741
Non-trainable params: 256
_________________________________________________________________
```

In [28]:
```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```



In [29]:
```
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()
```



In [30]:
```
# 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[30]:

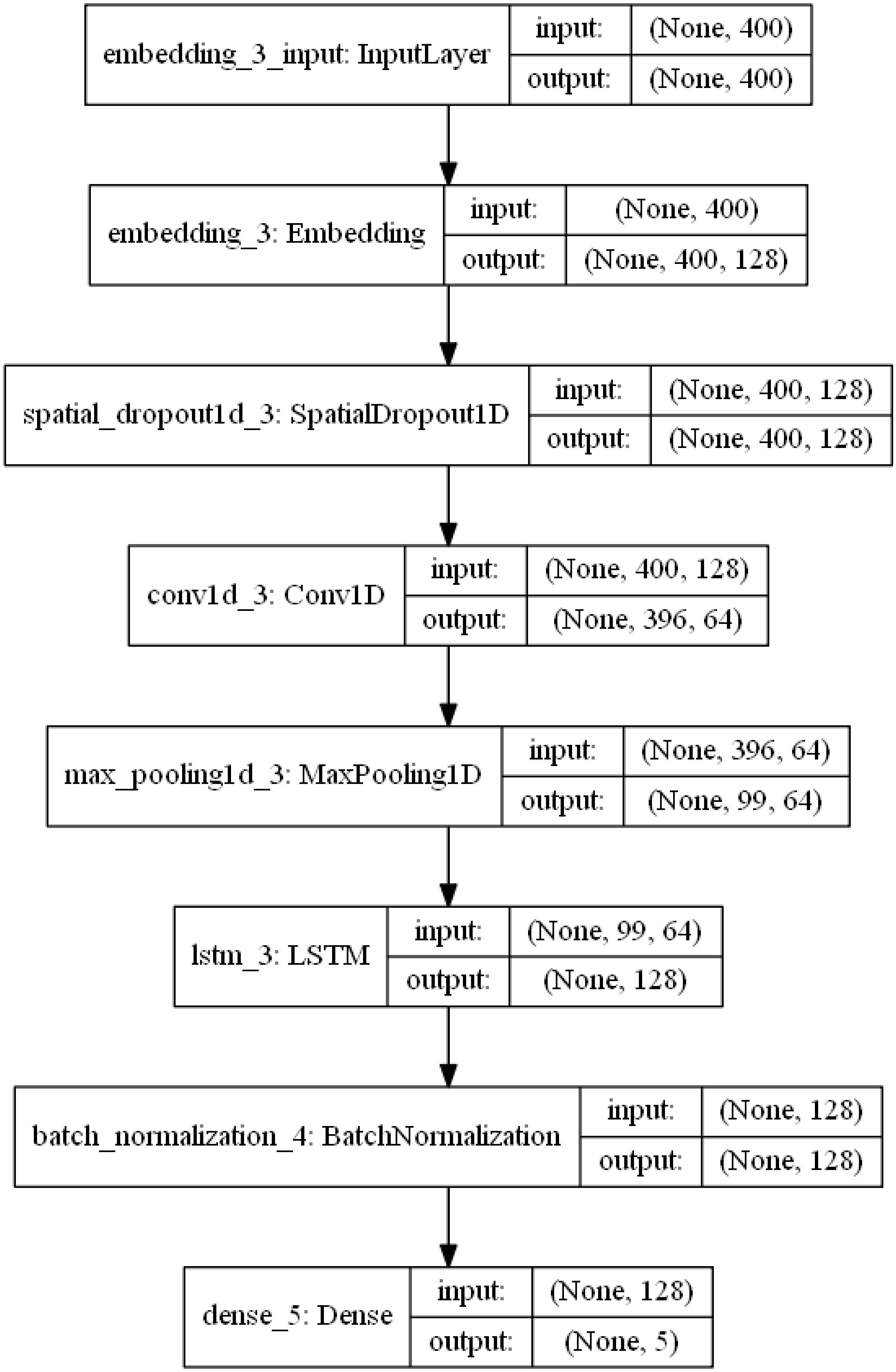
	1	2	3	4	5
1	35453	4833	1256	612	1291
2	2284	4190	2657	655	273
3	217	869	2544	1524	355
4	185	457	2710	8991	5068
5	748	394	1096	9979	71434

In [31]:
```
print(classification_report(y_pred_true, y_test_true))
```

```
              precision    recall  f1-score   support

           1       0.91      0.82      0.86     43445
           2       0.39      0.42      0.40     10059
           3       0.25      0.46      0.32      5509
           4       0.41      0.52      0.46     17411
           5       0.91      0.85      0.88     83651

    accuracy                           0.77    160075
   macro avg       0.57      0.61      0.59    160075
weighted avg       0.80      0.77      0.78    160075
```

In [32]:
```
plot_model(lstm, to_file='baseline.png', show_shapes=True)
```

Out[32]:



Let's save this model as well.

In [33]:
```
# 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 [34]:
```
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 [36]:
```
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', '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 [==============================] - 65s 217us/step - loss: 0.252
5 - accuracy: 0.9042 - mean_absolute_error: 0.1446 - val_loss: 0.6243 - val_a
ccuracy: 0.7587 - val_mean_absolute_error: 0.2416
Epoch 2/2
298804/298804 [==============================] - 64s 215us/step - loss: 0.180
8 - accuracy: 0.9319 - mean_absolute_error: 0.0989 - val_loss: 0.2134 - val_a
ccuracy: 0.9101 - val_mean_absolute_error: 0.1188
2
Train on 298804 samples, validate on 74702 samples
Epoch 1/3
298804/298804 [==============================] - 65s 217us/step - loss: 0.259
5 - accuracy: 0.9193 - mean_absolute_error: 0.1544 - val_loss: 0.2702 - val_a
ccuracy: 0.9323 - val_mean_absolute_error: 0.0871
Epoch 2/3
298804/298804 [==============================] - 64s 213us/step - loss: 0.184
6 - accuracy: 0.9354 - mean_absolute_error: 0.0984 - val_loss: 0.2115 - val_a
ccuracy: 0.9327 - val_mean_absolute_error: 0.0975
Epoch 3/3
298804/298804 [==============================] - 64s 213us/step - loss: 0.163
8 - accuracy: 0.9413 - mean_absolute_error: 0.0880 - val_loss: 0.2052 - val_a
ccuracy: 0.9335 - val_mean_absolute_error: 0.0906
3
Train on 298804 samples, validate on 74702 samples
Epoch 1/3
298804/298804 [==============================] - 64s 214us/step - loss: 0.261
1 - accuracy: 0.9195 - mean_absolute_error: 0.1575 - val_loss: 0.2571 - val_a
ccuracy: 0.9363 - val_mean_absolute_error: 0.0839
Epoch 2/3
298804/298804 [==============================] - 64s 213us/step - loss: 0.178
5 - accuracy: 0.9397 - mean_absolute_error: 0.0935 - val_loss: 0.2319 - val_a
ccuracy: 0.9363 - val_mean_absolute_error: 0.0749
Epoch 3/3
298804/298804 [==============================] - 64s 213us/step - loss: 0.157
9 - accuracy: 0.9452 - mean_absolute_error: 0.0832 - val_loss: 0.1909 - val_a
ccuracy: 0.9371 - val_mean_absolute_error: 0.0936
4
Train on 298804 samples, validate on 74702 samples
Epoch 1/3
298804/298804 [==============================] - 64s 213us/step - loss: 0.365
3 - accuracy: 0.8524 - mean_absolute_error: 0.2231 - val_loss: 0.4124 - val_a
ccuracy: 0.8639 - val_mean_absolute_error: 0.1834
Epoch 2/3
298804/298804 [==============================] - 63s 210us/step - loss: 0.306
2 - accuracy: 0.8735 - mean_absolute_error: 0.1799 - val_loss: 0.3641 - val_a
ccuracy: 0.8639 - val_mean_absolute_error: 0.1642
Epoch 3/3
298804/298804 [==============================] - 63s 212us/step - loss: 0.283
2 - accuracy: 0.8833 - mean_absolute_error: 0.1661 - val_loss: 0.3250 - val_a
ccuracy: 0.8684 - val_mean_absolute_error: 0.1794
5
Train on 298804 samples, validate on 74702 samples
Epoch 1/4
298804/298804 [==============================] - 64s 213us/step - loss: 0.350
9 - accuracy: 0.8569 - mean_absolute_error: 0.2055 - val_loss: 0.5123 - val_a
```

```
ccuracy: 0.7919 - val_mean_absolute_error: 0.3707
Epoch 2/4
298804/298804 [==============================] - 63s 211us/step - loss: 0.296
5 - accuracy: 0.8799 - mean_absolute_error: 0.1729 - val_loss: 0.3106 - val_a
ccuracy: 0.8755 - val_mean_absolute_error: 0.2020
Epoch 3/4
298804/298804 [==============================] - 63s 211us/step - loss: 0.269
3 - accuracy: 0.8915 - mean_absolute_error: 0.1565 - val_loss: 0.3045 - val_a
ccuracy: 0.8750 - val_mean_absolute_error: 0.1665
Epoch 4/4
298804/298804 [==============================] - 63s 209us/step - loss: 0.246
0 - accuracy: 0.9032 - mean_absolute_error: 0.1411 - val_loss: 0.3161 - val_a
ccuracy: 0.8738 - val_mean_absolute_error: 0.1611
```

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

Testing

In [37]:

```
%%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
0.3393659222239575
0.7590816804622833
Wall time: 5min 32s
```

In [38]:

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

Out[38]:

	1	2	3	4	5
1	34636	4825	1310	496	780
2	1540	2489	1026	269	140
3	768	1658	3089	1376	415
4	623	1042	3159	7653	3443
5	1320	729	1679	11967	73643

In [39]:
```
print(classification_report(ps["pred"], y_test_true[0]))
```

```
              precision    recall  f1-score   support

           1       0.89      0.82      0.86     42047
           2       0.23      0.46      0.31      5464
           3       0.30      0.42      0.35      7306
           4       0.35      0.48      0.41     15920
           5       0.94      0.82      0.88     89338

    accuracy                           0.76    160075
   macro avg       0.54      0.60      0.56    160075
weighted avg       0.81      0.76      0.78    160075
```

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 [40]:
```
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 [41]:
```
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 [42]:
```
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))])
```

```
[0.3218366390754334, 0.7678338278931751]
```

In [43]:
```
# 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[43]:

	1	2	3	4	5
1	36044	5513	1733	767	1157
2	1452	3157	1747	508	201
3	259	1053	2776	1420	366
4	204	500	2658	7995	3758
5	928	520	1349	11071	72939

In [44]:
```
print(classification_report(y_pred_true, y_test_true))
```

```
              precision    recall  f1-score   support

           1       0.91      0.82      0.86     43445
           2       0.39      0.42      0.40     10059
           3       0.25      0.46      0.32      5509
           4       0.41      0.52      0.46     17411
           5       0.91      0.85      0.88     83651

    accuracy                           0.77    160075
   macro avg       0.57      0.61      0.59    160075
weighted avg       0.80      0.77      0.78    160075
```

Challenges

Challenge 5

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

```
(500, 3)
```

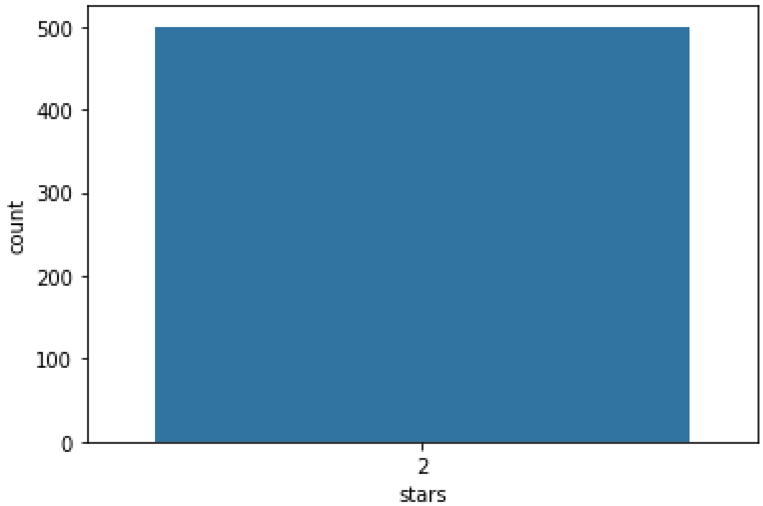
Out[45]:

	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 0x22766065048>



Pre-processing

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

Out[47]:

	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 [48]:
```
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 [49]:
```
# # 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 [50]:
```
# 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 80us/step
[2.0360392417907716, 0.2759999930858612, 0.29150819778442383]
500/500 [==============================] - 0s 552us/step
[1.3482585124969482, 0.39800000190734863, 0.26108863949775696]
[1.034, 0.226]
```

Attempt Ensemble

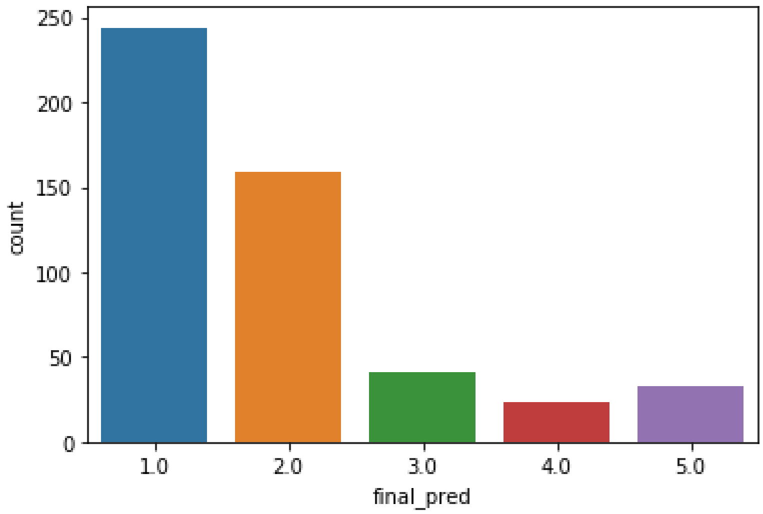
In [51]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# 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))])
```

```
[0.86, 0.318]
```

Misc.

In [52]:
```
sns.countplot(all_preds["final_pred"])
```

Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x22a5cd48e48>



Challenge 6

In [53]:
```
c6 = pd.read_json("./yelp_challenge_6_with_answers.jsonl", lines = True)
print(c6.shape)
c6.head()
```

```
(500, 3)
```

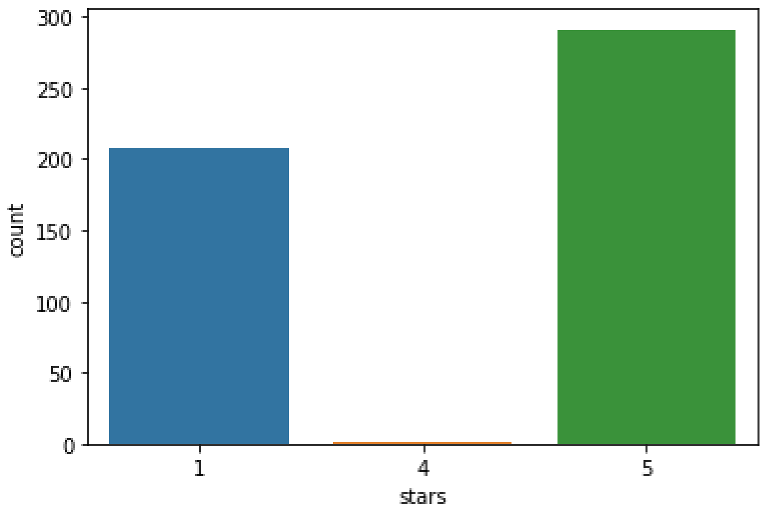
Out[53]:

	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 [54]:
```
sns.countplot(c6['stars'])
```

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x22a5cda49c8>



Pre-processing

In [55]:
```
c6['text'] = c6['text'].apply(clean_text)
c6.head()
```

Out[55]:

	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 no intent visit place disgust next door...	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 [56]:
```
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 [57]:
```
# 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 88us/step
[2.419058826446533, 0.42800000309944153, 0.2563921809196472]
500/500 [==============================] - 0s 506us/step
[2.2065195159912108, 0.42800000309944153, 0.25307539105415344]
[2.03, 0.42]
```

Attempt Ensemble

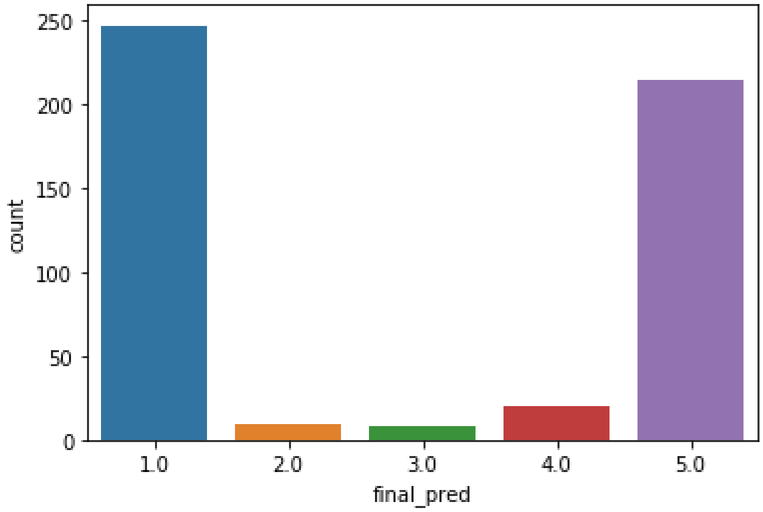
In [58]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# 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))])
```

```
[2.05, 0.45]
```

Misc.

In [59]:
```
sns.countplot(all_preds["final_pred"])
```

Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x22a5cf96688>



Challenge 3

In [60]:
```
c3 = pd.read_json("./yelp_challenge_3_with_answers.jsonl", lines = True)
print(c3.shape)
c3.head()
```

(534, 3)

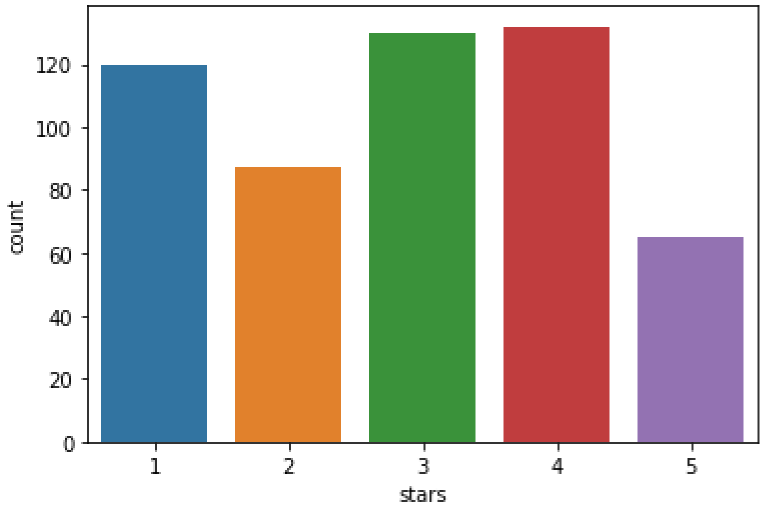
Out[60]:

	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 [61]:
```
sns.countplot(c3['stars'])
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x22a5cf58688>



Pre-processing

In [62]:
```
c3['text'] = c3['text'].apply(clean_text)
c3.head()
```

Out[62]:

	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	veri bad food avoid group 4 veri hungri came o...	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 [63]:
```
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 [64]:
```
# 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 82us/step
[1.2328123662355688, 0.5393258333206177, 0.20918305218219757]
534/534 [==============================] - 0s 536us/step
[0.9241146900680628, 0.5898876190185547, 0.2071395367383957]
[0.5805243445692884, 0.5318352059925093]
```

Attempt Ensemble

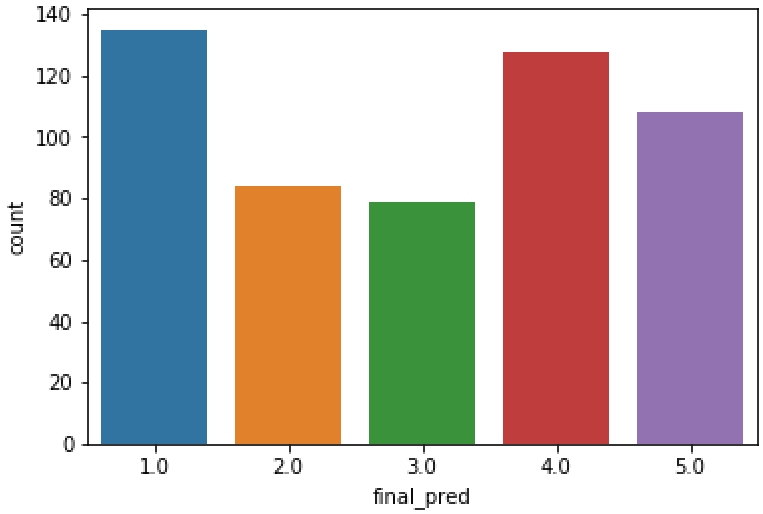
In [65]:
```
# Baseline
baseline_preds = pd.DataFrame(baseline.predict(X_baseline), columns=cols)
baseline_preds['baseline_pred'] = baseline_preds.idxmax(axis=1)

# 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))])
```

```
[0.4850187265917603, 0.5880149812734082]
```

Misc.

In [66]:
```
sns.countplot(all_preds["final_pred"])
```

Out[66]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22a5cefe488>
```



Challenge 8

In [67]:
```
c8 = pd.read_json("./yelp_challenge_8_with_answers.jsonl", lines = True)
print(c8.shape)
c8.head()
```

```
(500, 3)
```

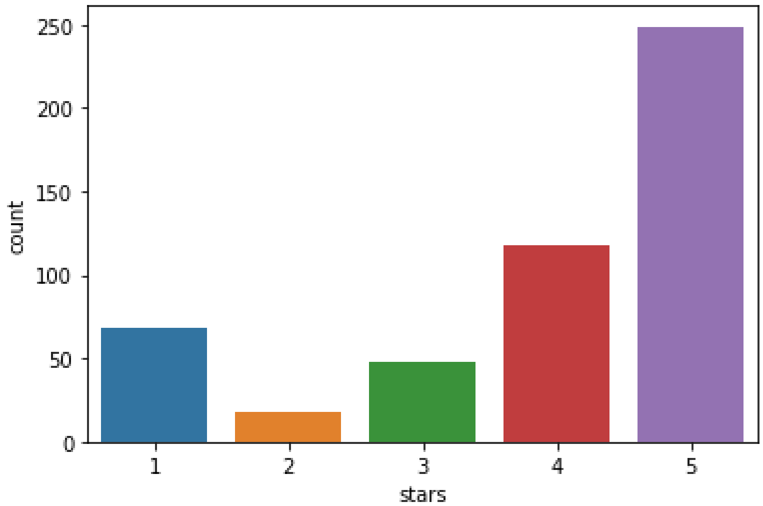
Out[67]:

	review_id	text	stars
0	qOOv-A-vo3kMT0yi4jIlIg	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	fAhp8lwuGNT0ywKmsCs6VQ	Very old and dirty vans.	1

Quick EDA

In [68]:
```
sns.countplot(c8['stars'])
```

Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x22a5d0384c8>



Pre-processing

In [69]:
```
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[69]:

	review_id	text	stars
0	qOOv-A-vo3kMT0yi4jIllg	not 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	veri old dirti van	1

Load previous tokenizer

In [70]:
```
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 [71]:

```
# 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
[1.060733362674713, 0.6320000290870667, 0.19019262492656708]
500/500 [==============================] - 0s 506us/step
[0.9087534489631652, 0.6420000195503235, 0.18208713829517365]
[0.564, 0.624]
```

Attempt Ensemble

In [72]:

```
# 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))])
```

```
[0.544, 0.644]
```

Misc.

In [73]:
```
sns.countplot(all_preds["final_pred"])
```

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x22a4cb71948>