



Athens University of Economics and Business

Department of Management Science & Technology
- MSc in Business Analytics

Big Data Content Analytics

‘Sentiment analysis on restaurant reviews from yelp website’



Students:

Alexandros Ntzoufas

Romanos Oikonomidis

Sofia Zgkouri

September 2019

Code repository: <https://github.com/antzoufas/AI>

1. Task description

Sentiment analysis is the automated process of understanding an opinion about a given subject from written or spoken language. Currently, more than 2.5 quintillion bytes of data are generated in a daily basis worldwide, sentiment analysis has become a key tool for making sense of that data. This has allowed companies to understand the social sentiment of their brand, product or service while monitoring online conversations in order to get key insights and automate all kind of processes. With the recent advances in deep learning, the ability of algorithms to analyse text has improved considerably. Creative use of advanced artificial intelligence techniques can be an effective tool for doing in-depth research.

The main aim of this assignment is to identify whether reviews of the customers for hotels, restaurants, coffee shops, etc are negative, positive or neutral and if there is place for improvement. More specifically, it can be analyzed if the restaurants of a specific area gather positive or negative reviews, so another competitor can offer better services and gain higher share of the market. Another example is to identify if the restaurants of specific island in Greece (i.e. Santorini) that serve Mediterranean cuisine collect negative or possitive reviews by the visitors, so modifications in the sector should be implemented.

2. Data

The dataset under study is containing reviews from the Yelp website (User Reviews and Recommendations of Best Restaurants, Shopping, Nightlife, Food, Entertainment, Things to Do, Services and More) was downloaded from Kaggle : <https://www.kaggle.com/yelp-dataset/yelp-dataset>

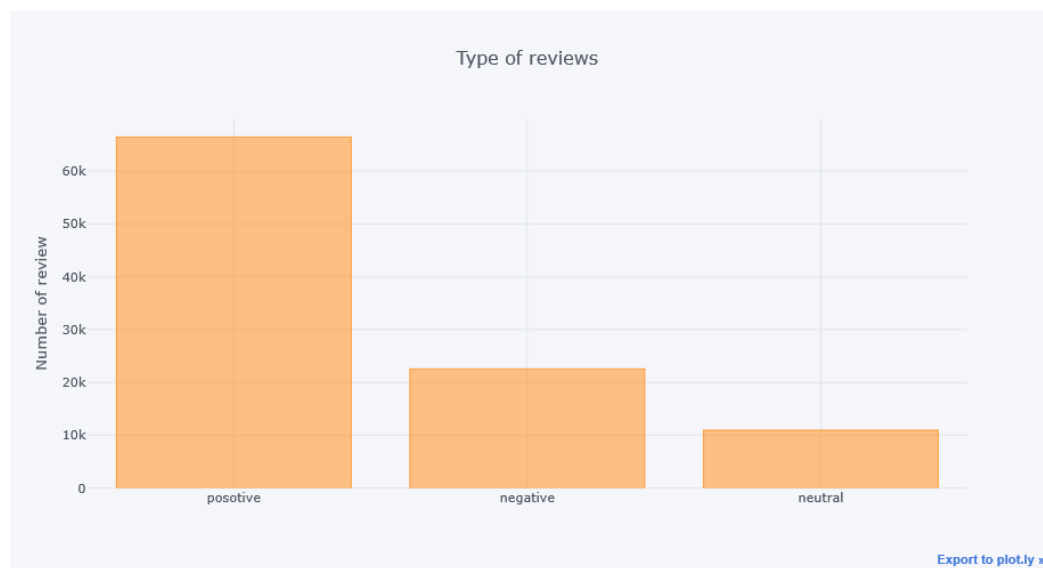
Sentiment analysis or in other words, opinion mining was performed on this data using Recurrent Neural Networks and Convolution Neural Networks.

From the initial dataset downloaded from Kaggle, of size ~5GB, a chunk dataset of ~800MB was handled in python. The initial file was split using Gitbash terminal commands (split <data.json> -n 6). From this cut file, 100.000 rows were used as an input to the model. The data were in json format.

The categoriasation of a review was done by taking into consideration the ratings of the users (stars column of the dataset). Reviews with 1 or 2 stars are considered as negative, reviews with 3 stars as neutral while reviews with 4 or 5 stars as positive. An additional column was created in the dataset with labels 0,1,2 for representing the 3 levels of characterization of a review (negative, neutral, positive for 0,1 and 2 respectively).

	review_id	user_id	stars	text	sentiment
0	Q1sbwvVQXV2734tPgoKj4Q	hG7b0MtEbXx5QzbzE6C_VA	1.0	Total bill for this horrible service? Over \$8G...	0
1	GJXCdrto3ASJOqKeVWPi6Q	yXQM5uF2jS6es16SjzNHfg	5.0	I *adore* Travis at the Hard Rock's new Kelly ...	2
2	2TzJjDVDEuAW6MR5Vuc1ug	n6-Gk65cPZL6Uz8qRm3NYw	5.0	I have to say that this office really has it t...	2
3	yi0R0Ugj_xUx_Nek0-Qig	dacAlZ6fTM6mqwW5uxkskg	5.0	Went in for a lunch. Steak sandwich was delici...	2
4	11a8sVPMUFTaC7_ABRkmtw	ssoyf2_x0EQMed6fgHeMyQ	1.0	Today was my second out of three sessions I ha...	0

The columns review id, user id and stars were removed from the initial data. Then the dataset was split into train and test dataset, with an analogy of 90% - 10%. As we see from the following figure, the majority of the reviews are positive. Almost the one fourth of the reviews are negative while only the one tenth is neutral.



3. Text Classification Using Convolutional Neural Network (CNN) :

Preprocessing

The text preprocessing that we followed included:

- Converting all text to lower case.
- Replace symbols by space in text.
- Delete symbols from text.
- Delete stop words.

- Delete digits in text.

Then during the LSTM modeling, we vectorized consumers complaints text, by turning each text into either a sequence of integers or into a vector. Then we limited the data set (vocabulary) to the top 5,000 words and we set the max number of words in each complaint to 250.

```
In [48]: from nltk.corpus import stopwords

df = df.reset_index(drop=True)
replace_by_space = re.compile('[/(){}\\[\\]\\|@,;:]')
exclude_bad_symbols = re.compile('[^0-9a-z #+_]')
stopwords = set(stopwords.words('english'))

def clean_text(text):
    """
    text: a string
    return: modified initial string
    """
    text = text.lower() # lowercase text
    text = replace_by_space.sub(' ', text) # replace by space
    text = exclude_bad_symbols.sub('', text) # remove symbols
    text = text.replace('x', '')
    # text = re.sub(r'\W+', '', text)
    text = ' '.join(word for word in text.split() if word not in stopwords) # remove stopwords
    return text
df['text'] = df['text'].apply(clean_text)
df['text'] = df['text'].str.replace('\d+', '')
```

Then we used the `pad_sequences` is used to ensure that all sequences in a list have the same length. Furthermore, since we have three different values in the categorical variable, negative positive and neutral, we converted the categorical labels to numbers.

```
In [3]: from keras.preprocessing.text import Tokenizer
# The maximum number of words to be used. (most frequent)
MAX_NB_WORDS = 50000
# Max number of words in each complaint.
MAX_SEQUENCE_LENGTH = 250
# This is fixed.
EMBEDDING_DIM = 100
tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(df['text'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

Using TensorFlow backend.

Found 137484 unique tokens.

```
In [4]: from keras.preprocessing.sequence import pad_sequences

X = tokenizer.texts_to_sequences(df['text'].values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)
```

Shape of data tensor: (100000, 250)

```
In [5]: Y = pd.get_dummies(df['sentiment']).values
print('Shape of label tensor:', Y.shape)
```

Shape of label tensor: (100000, 3)

```
In [6]: import pandas as pd
Y = pd.get_dummies(df['sentiment']).values
print('Shape of label tensor:', Y.shape)
```

Shape of label tensor: (100000, 3)

With the `sklearn train_test_split` function, we splitted the dataset to train and test, with proportions, 90% and 10% respectively.

```
In [7]: from sklearn.model_selection import train_test_split
import numpy as np
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.10, random_state = 42)
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)

(90000, 250) (90000, 3)
(10000, 250) (10000, 3)
```

Long Short Term Memory networks , just called “LSTMs” are a special kind of RNN, capable of learning long-term dependencies. The model that we built includes:

- Firstly we added the first layer which is a Sequential model which needs to receive information about its input shape.
- Then we added an embedded layer with 100 length vectors to represent each word.
- Then we added a SpatialDropout1D performs variational dropout in NLP models. This version performs the same function as Dropout, however it drops entire 1D feature maps instead of individual elements
- Then we added the next layer is the LSTM layer with 100 memory units.
- The output layer must create 3 output values, one for each class for our labels.
- Since we have a multi class classification problem we added softmax as activation function.
- Finally, since we have a multi-class classification problem, categorical_crossentropy is used as the loss function.

```
In [8]: from keras.models import Sequential
from keras.layers import Embedding
from keras import layers
from keras.layers.core import Dense, SpatialDropout1D
from keras.layers.recurrent import LSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping

model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```


WARNING:tensorflow:From C:\Users\Alexandros Ntzoufas\Anaconda3\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\Alexandros Ntzoufas\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use 'rate' instead of 'keep_prob'. Rate should be set to 'rate = 1 - keep_prob'.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 250, 100)	5000000
spatial_dropout1d_1 (Spatial	(None, 250, 100)	0
lstm_1 (LSTM)	(None, 100)	80400
dense_1 (Dense)	(None, 3)	303
Total params: 5,080,703		
Trainable params: 5,080,703		
Non-trainable params: 0		

We train our model for 15 epochs, with an EarlyStopping, patience=5 and we yield the following results.

```
In [10]: epochs = 15
batch_size = 64

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_split=0.1, callbacks=[EarlyStopping(monitor
```

<  >

Train on 81000 samples, validate on 9000 samples

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1/15	0.4412	0.8350	0.3815	0.8503
2/15	0.3409	0.8694	0.3849	0.8480
3/15	0.2841	0.8923	0.4045	0.8492
4/15	0.2391	0.9092	0.4268	0.8422
5/15	0.1975	0.9260	0.4860	0.8403
6/15	0.1676	0.9380	0.5178	0.8353

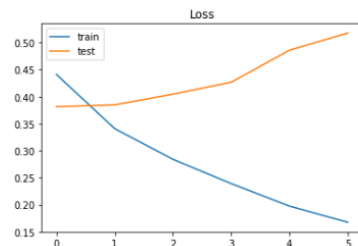
The accuracy of the model in the training dataset is 95.17% while for the testing set is 83.34%.

```
In [13]: loss, accuracy = model.evaluate(X_train, Y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, Y_test, verbose=False)
print("Testing Accuracy: {:.4f}".format(accuracy))
```

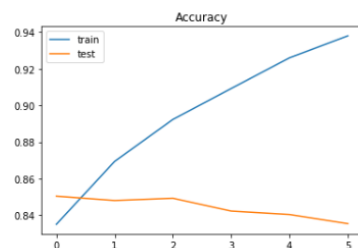
Training Accuracy: 0.9517
Testing Accuracy: 0.8334

As it arises from the following figure, our model tends to overfit after the 6th epoch.

```
In [14]: plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show();
```



```
In [15]: plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show();
```



The following matrices indicate the performance of the model. More specifically:

- Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. We see that our model has better results for the negative and positive labels (0 and 2, respectively) than for neutral (1)
- Recall is the ratio of correctly predicted positive observations to the all observations in actual class. Again, our model performs better for negative and positive reviews.
- F1 score - F1 Score is the weighted average of Precision and Recall. The same results applies for F1-Score as well.

```
In [16]: from sklearn.metrics import confusion_matrix, classification_report
```

```
In [17]: Y_test_pred = model.predict_classes(X_test)
conf_mat = confusion_matrix(Y_test.argmax(axis=1),
                             Y_test_pred)

pd.DataFrame(conf_mat)
```

```
Out[17]:
```

	0	1	2
0	1726	257	211
1	214	405	497
2	174	313	6203

```
In [18]: print(classification_report(Y_test.argmax(axis=1),
                                     Y_test_pred,
                                     digits=4))
```

	precision	recall	f1-score	support
0	0.8165	0.7867	0.8013	2194
1	0.4154	0.3629	0.3874	1116
2	0.8976	0.9272	0.9121	6690
micro avg	0.8334	0.8334	0.8334	10000
macro avg	0.7098	0.6923	0.7003	10000
weighted avg	0.8260	0.8334	0.8293	10000

4. Text Classification Using Convolutional Neural Network (CNN)

CNN is a class of deep, feed-forward artificial neural networks and use a variation of multilayer perceptrons designed to require minimal preprocessing. However CNNs are generally used in computer vision, they have been applied to various NLP tasks as well. In our case we applied the following CNN in our model in order to see the results and compare them with the other models.

More specifically, we applied the following model with a Sequential layer and an embedded layer with 100 length vectors. Then we added a Conv1D layer which creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal). Then we added a GlobalMaxPooling1D layer for temporal data which takes the max vector over the steps dimension

Then we added a ReLU activation function to the output to introduce nonlinearities into the model. Since we have a multi class classification problem we added softmax as activation function. Finally, since we have a multi-class classification problem, categorical_crossentropy is used as the loss function.

```
In [9]: from keras.layers import Activation, Dense
from keras.models import Sequential
from keras import layers
vocab_size = len(tokenizer.word_index) + 1
maxlen = 100

embedding_dim = 100

model = Sequential()
model.add(layers.Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(3, activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

WARNING:tensorflow:From C:\Users\Alexandros Ntzoufas\Anaconda3\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 250, 100)	5000000
conv1d_1 (Conv1D)	(None, 246, 128)	64128
global_max_pooling1d_1 (Glob	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290
dense_2 (Dense)	(None, 3)	33
Total params: 5,065,451		
Trainable params: 5,065,451		
Non-trainable params: 0		

As we see from the following figure, we fitted the model for 15 epochs, with an EarlyStopping with patience=5. The model trained for 6 epochs, with an accuracy of 83.1% and loss of 0.919.

```
In [13]: from keras.models import Sequential
from keras.layers import Embedding
from keras import layers
from keras.layers.core import Dense, SpatialDropout1D
from keras.layers.recurrent import LSTM
from keras.callbacks import ModelCheckpoint, EarlyStopping

epochs = 15
batch_size = 64

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_split=0.1, callbacks=[EarlyStopping(monitor
```

```
<
Train on 81000 samples, validate on 9000 samples
Epoch 1/15
81000/81000 [=====] - 246s 3ms/step - loss: 0.4065 - acc: 0.8430 - val_loss: 0.3717 - val_acc: 0.8524
Epoch 2/15
81000/81000 [=====] - 246s 3ms/step - loss: 0.2522 - acc: 0.9038 - val_loss: 0.4215 - val_acc: 0.8329
Epoch 3/15
81000/81000 [=====] - 239s 3ms/step - loss: 0.1194 - acc: 0.9597 - val_loss: 0.4982 - val_acc: 0.8436
Epoch 4/15
81000/81000 [=====] - 241s 3ms/step - loss: 0.0429 - acc: 0.9879 - val_loss: 0.6410 - val_acc: 0.8268
Epoch 5/15
81000/81000 [=====] - 236s 3ms/step - loss: 0.0130 - acc: 0.9972 - val_loss: 0.7701 - val_acc: 0.8381
Epoch 6/15
81000/81000 [=====] - 240s 3ms/step - loss: 0.0045 - acc: 0.9993 - val_loss: 0.8991 - val_acc: 0.8306
```

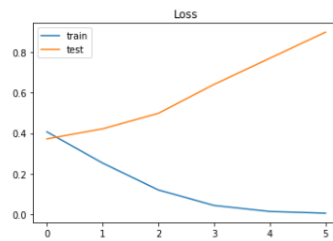
```
In [16]: accr = model.evaluate(X_test, Y_test)
print('Test set\n Loss: {:.3f}\n Accuracy: {:.3f}'.format(accr[0], accr[1]))

10000/10000 [=====] - 7s 731us/step
Test set
Loss: 0.919
Accuracy: 0.831
```

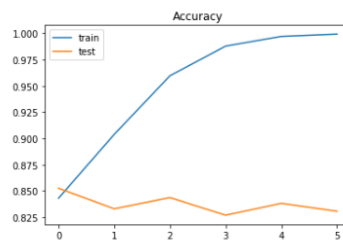
```
In [17]: plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show();
```

As we see from the following figure, there is an over fitting in the training dataset.


```
In [17]: plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show();
```



```
In [18]: plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show();
```



With regards to the performance in the training dataset, we can see that the model can predict with a high accuracy the positive and the negative reviews; however the probability to predict the neutral reviews is less than 50%. This may happens since the number of the neutral reviews were low in number.

```
In [19]: from sklearn.metrics import confusion_matrix, classification_report
Y_test_pred = model.predict_classes(X_test)
conf_mat = confusion_matrix(Y_test.argmax(axis=1),
                             Y_test_pred)
pd.DataFrame(conf_mat)
```

```
Out[19]:
```

	0	1	2
0	1695	297	202
1	199	481	436
2	135	419	6136

```
In [21]: print(classification_report(Y_test.argmax(axis=1),
                                     Y_test_pred,
                                     digits=4))
```

	precision	recall	f1-score	support
0	0.8354	0.7726	0.8027	2194
1	0.4018	0.4310	0.4159	1116
2	0.9058	0.9172	0.9115	6690
micro avg	0.8312	0.8312	0.8312	10000
macro avg	0.7143	0.7069	0.7100	10000
weighted avg	0.8341	0.8312	0.8323	10000

5. Text Classification BOW (Bag of Words) with Keras

The bag-of-words model is a simplifying representation used in natural language processing and information retrieval (IR). More specifically, we used tokenizer methods to count the unique words in our vocabulary. Then we feed a one-hot vector to our model and we built our text classification model as follows:

```
train_size = int(len(df) * .7)
train_posts = df['text'][:train_size]
train_tags = df['sentiment'][:train_size]

test_posts = df['text'][train_size:]
test_tags = df['sentiment'][train_size:]

max_words = 10000
tokenizer = text.Tokenizer(num_words=max_words, char_level=False)
tokenizer.fit_on_texts(train_posts) # only fit on train

x_train = tokenizer.texts_to_matrix(train_posts)
x_test = tokenizer.texts_to_matrix(test_posts)

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

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)

batch_size = 32

# Build the model
model = Sequential()
model.add(Dense(512, input_shape=(max_words,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

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

model.summary()
```

Layer (type)	Output Shape	Param #
dense_13 (Dense)	(None, 512)	5120512
activation_13 (Activation)	(None, 512)	0
dropout_7 (Dropout)	(None, 512)	0
dense_14 (Dense)	(None, 3)	1539
activation_14 (Activation)	(None, 3)	0
Total params: 5,122,051		
Trainable params: 5,122,051		
Non-trainable params: 0		

- Firstly we added the first layer which is a Sequential
- Then we added an Dense layer of dense 512-dimensional vectors
- Then we added a a ReLU activation function , a Dropout layer, and again a Dense layer
- Since we have a multi class classification problem we added softmax as activation function.
- Finally, since we have a multi-class classification problem, categorical_crossentropy is used as the loss function.

```
In [55]: from keras.callbacks import ModelCheckpoint, EarlyStopping
epochs = 15
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.1, callbacks=[EarlyStopping(monitor='val_loss', patience=5, min_delta=0.0001)])
```

Train on 63000 samples, validate on 7000 samples

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1/15	0.4284	0.8380	0.3821	0.8539
2/15	0.3022	0.8867	0.4091	0.8437
3/15	0.1922	0.9328	0.4570	0.8453
4/15	0.1029	0.9675	0.5397	0.8397
5/15	0.0540	0.9930	0.5545	0.8420

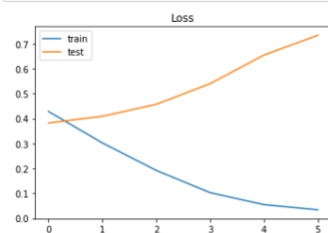
As we see from the following figure, we fitted the model for 15 epochs, with an EarlyStopping with patience=5. The model trained for 6 epochs, with an accuracy of 98.3% in the training set and 84% in the test set.

```
In [56]: loss, accuracy = model.evaluate(x_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(x_test, y_test, verbose=False)
print("Testing Accuracy: {:.4f}".format(accuracy))
```

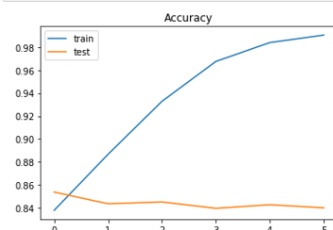
Training Accuracy: 0.9834
Testing Accuracy: 0.8400

As we see from the following figure, there is an over fitting in the training dataset close to the 6th epoch.

```
In [57]: plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show();
```



```
In [58]: plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show();
```



Considering the performance in the training dataset, we can see that the model predicts with a high accuracy the positive and the negative reviews; however the probability to predict the neutral reviews is less than 50%. This may happen since

the number of the neutral reviews were low in number.

```
In [61]: from sklearn.metrics import confusion_matrix, classification_report
Y_test_pred = model.predict_classes(x_test)
conf_mat = confusion_matrix(y_test.argmax(axis=1),
                             Y_test_pred)
pd.DataFrame(conf_mat)
```

```
Out[61]:
```

	0	1	2
0	5594	518	724
1	871	822	1620
2	520	546	18785

```
In [62]: print(classification_report(y_test.argmax(axis=1),
                                     Y_test_pred,
                                     digits=4))
```

	precision	recall	f1-score	support
0	0.8009	0.8183	0.8095	6836
1	0.4358	0.2481	0.3162	3313
2	0.8891	0.9463	0.9168	19851
micro avg	0.8400	0.8400	0.8400	30000
macro avg	0.7086	0.6709	0.6888	30000
weighted avg	0.8189	0.8400	0.8260	30000

6. Conclusions

In the context of this assignment, various models were tested in order to identify whether the sentiment polarity of the reviews of customers can be predicted. More specifically, the yelp dataset was processed and tested with LSTM RNN model, a CNN model and and a BOW with Keras model. The scope of this analysis is to identify the positive and negative reviews of consumers for specific sectors of interests (restaurants, hotels, coffee shops) and specific areas of interest. With this process, we can identify sectors or places where potential for improvement or investments can be initiated.

As it is presented in the following table, all models brought relevant results. The models brought good prediction with regards to the positive and negative reviews of the customers; however the neutral reviews were more difficult to be identified.

Accuracy(%)	Train dataset	Test dataset
LSTM RNN	95.17%	83.34%
CNN	98.3%	83.6%
BOW with KERAS	98.3%	84%