In [1]:

```
# Create dataframes
import pandas as pd
import warnings

# amazon
amazon = open('C:/Users/hkeim/OneDrive/Documents/School/D213/Task Two/sentiment labelled sent

a_labels, a_texts = [], []
for i, line in enumerate(amazon.split('\n')):
    content = line.split('\t')
    if len(content) > 1:
        a_texts.append(content[0])
        a_labels.append(content[1])

df_a = pd.DataFrame()
df_a['label'] = a_labels
df_a['text'] = a_texts

# imdb
imdb = open('C:/Users/hkeim/OneDrive/Documents/School/D213/Task Two/sentiment labelled senten

i_labels, i_texts = [], []
for i, line in enumerate(imdb.split('\n')):
    content = line.split('\t')
    if len(content) > 1:
        i_texts.append(content[0])
        i_labels.append(content[1])

df_i = pd.DataFrame()
df_i['label'] = i_labels
df_i['text'] = i_texts

# yelp
yelp = open('C:/Users/hkeim/OneDrive/Documents/School/D213/Task Two/sentiment labelled senten

y_labels, y_texts = [], []
for i, line in enumerate(yelp.split('\n')):
    content = line.split('\t')
    if len(content) > 1:
        y_texts.append(content[0])
        y_labels.append(content[1])

df_y = pd.DataFrame()
df_y['label'] = y_labels
df_y['text'] = y_texts

# Combine site dataframes
df = pd.concat([df_a, df_i, df_y], ignore_index=True)
df.label = df.label.astype(int)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   label   3000 non-null   int32 
 1   text    3000 non-null   object
dtypes: int32(1), object(1)
memory usage: 35.3+ KB
```

In [2]:
```
# Extract characters from reviews
characters = df['text']
list_of_characters = []
for text in characters:
    for character in text:
        if character not in list_of_characters:
            list_of_characters.append(character)
print(list_of_characters)
```

```
['S', 'o', ' ', 't', 'h', 'e', 'r', 'i', 's', 'n', 'w', 'a', 'y', 'f', 'm', 'p', 'l', 'u',
'g', 'U', 'I', 'b', 'c', 'v', '.', 'G', 'd', ',', 'E', 'x', 'j', 'T', '4', '5', 'M', 'A', 'J',
'O', 'R', 'P', 'B', 'L', '!', 'z', 'N', 'W', 'q', 'H', '+', 'V', '"', 'Y', 'D', 'F', 'k', "'",
'K', 'C', '/', '7', '3', '6', '8', '0', '2', '?', 'Z', '-', '1', ':', ')', '(', 'Q', '&', '$',
'*', ';', 'X', '%', '9', '#', '[', ']', 'Â', '–', 'Ã', '©', '…', '¥', '—', 'ª']
```

In [3]:
```
# Text cleaning
from nltk.tokenize import word_tokenize
import numpy as np
import string

i = 0
df['clean'] = ''
for row in df.text:
    # split into words
    tokens = word_tokenize(row)
    # convert to lower case
    tokens = [token.lower() for token in tokens]
    # remove punctuation
    table = str.maketrans('', '', string.punctuation)
    words = [token.translate(table) for token in tokens]
    # remove non-alphabetic or numeric tokens
    words = [word for word in words if word.isalnum()]
    #print(words)
    df['clean'][i] = ' '.join(words)
    i += 1
df.clean =' ' + df.clean
df.head()
```

```
<ipython-input-3-9092d7aef1ef>:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/
indexing.html#returning-a-view-versus-a-copy
  df['clean'][i] = ' '.join(words)
```

Out[3]:

	label	text	clean
0	0	So there is no way for me to plug it in here i...	so there is no way for me to plug it in here ...
1	1	Good case, Excellent value.	good case excellent value
2	1	Great for the jawbone.	great for the jawbone
3	0	Tied to charger for conversations lasting more...	tied to charger for conversations lasting mor...
4	1	The mic is great.	the mic is great

In [4]:
```
#Splitting pd.Series to list
data = df['clean'].values.tolist()
print(data[:5])
```

```
[' so there is no way for me to plug it in here in the us unless i go by a converter', ' good
case excellent value', ' great for the jawbone', ' tied to charger for conversations lasting m
```

```
ore than 45 minutesmajor problems', ' the mic is great']
```

In [5]:

```
# sequence length
df['words'] = df.clean.apply(lambda x: len(x.split()))
df.words.describe()
```

Out[5]:

```
count    3000.000000
mean       12.038667
std         8.019182
min         1.000000
25%         6.000000
50%        10.000000
75%        16.000000
max        71.000000
Name: words, dtype: float64
```

In [6]:

```
# Create 2d numpy arrays
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(data)
sequences = tokenizer.texts_to_sequences(data)

# vocabulary size
print('Vocabulary Size:', len(tokenizer.word_index)+1)
```

```
Vocabulary Size: 5278
```

In [7]:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
sequences = pad_sequences(sequences, maxlen = 71)
print('padded sequence:', sequences[:1])
```

```
padded sequence: [[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0   29   42    6   59  115   12
    74    7  364    5   11   65   11    1  177  636    3   81   62    4
  2210]]
```

In [8]:

```
# Split data into test and train sets
from tensorflow import one_hot
x = np.array(sequences)
y = df.label.values

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 15,

y_train = one_hot(y_train, 2)
y_test = one_hot(y_test, 2)

# View sample sizes
print('training size:', x_train.shape)
print('testing size:', x_test.shape)
```

```
training size: (2400, 71)
testing size: (600, 71)
```

In [9]:

```
# Create/export test and train datasets
df_x_train = pd.DataFrame(x_train)
df_x_test = pd.DataFrame(x_test)
df_y_train = pd.DataFrame(y_train)
```

```
df_y_test = pd.DataFrame(y_test)

df_x_train.to_csv("C:/Users/hkeim/OneDrive/Documents/School/D213/Task Two/Keim D213 Task Two
df_x_test.to_csv("C:/Users/hkeim/OneDrive/Documents/School/D213/Task Two/Keim D213 Task Two x
df_y_train.to_csv("C:/Users/hkeim/OneDrive/Documents/School/D213/Task Two/Keim D213 Task Two
df_y_test.to_csv("C:/Users/hkeim/OneDrive/Documents/School/D213/Task Two/Keim D213 Task Two y
```

In [15]:

```
# Import Keras libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Embedding, GlobalAveragePooling1D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint


# Create model
early_stopping_monitor = EarlyStopping(patience = 2)

model = Sequential()

# Add layers
model.add(Embedding(input_dim = 5278, output_dim = 9, input_length = 71))
model.add(GlobalAveragePooling1D())
model.add(Dense(12, activation = 'relu'))
model.add(Dense(5, activation = 'relu'))
model.add(Dense(2, activation = 'sigmoid'))

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

model.summary()

history = model.fit(x_train, y_train, batch_size = 71, epochs=50 ,validation_data = (x_test,
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 71, 9)             47502
_________________________________________________________________
global_average_pooling1d_2 ( (None, 9)                 0
_________________________________________________________________
dense_6 (Dense)              (None, 12)                120
_________________________________________________________________
dense_7 (Dense)              (None, 5)                 65
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 12
=================================================================
Total params: 47,699
Trainable params: 47,699
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
34/34 [==============================] - 0s 6ms/step - loss: 0.6931 - accuracy: 0.5050 - val_l
oss: 0.6929 - val_accuracy: 0.5000
Epoch 2/50
34/34 [==============================] - 0s 2ms/step - loss: 0.6925 - accuracy: 0.5221 - val_l
oss: 0.6922 - val_accuracy: 0.6267
Epoch 3/50
34/34 [==============================] - 0s 2ms/step - loss: 0.6909 - accuracy: 0.5933 - val_l
oss: 0.6907 - val_accuracy: 0.5117
Epoch 4/50
34/34 [==============================] - 0s 2ms/step - loss: 0.6879 - accuracy: 0.5525 - val_l
oss: 0.6875 - val_accuracy: 0.5350
Epoch 5/50
34/34 [==============================] - 0s 2ms/step - loss: 0.6814 - accuracy: 0.6121 - val_l
```

```
oss: 0.6817 - val_accuracy: 0.5933
Epoch 6/50
34/34 [==============================] - 0s 2ms/step - loss: 0.6691 - accuracy: 0.6538 - val_l
oss: 0.6711 - val_accuracy: 0.5533
Epoch 7/50
34/34 [==============================] - 0s 2ms/step - loss: 0.6480 - accuracy: 0.7042 - val_l
oss: 0.6542 - val_accuracy: 0.6450
Epoch 8/50
34/34 [==============================] - 0s 2ms/step - loss: 0.6180 - accuracy: 0.7425 - val_l
oss: 0.6308 - val_accuracy: 0.7533
Epoch 9/50
34/34 [==============================] - 0s 2ms/step - loss: 0.5773 - accuracy: 0.8279 - val_l
oss: 0.6046 - val_accuracy: 0.7600
Epoch 10/50
34/34 [==============================] - 0s 2ms/step - loss: 0.5312 - accuracy: 0.8587 - val_l
oss: 0.5750 - val_accuracy: 0.8033
Epoch 11/50
34/34 [==============================] - 0s 2ms/step - loss: 0.4814 - accuracy: 0.8871 - val_l
oss: 0.5468 - val_accuracy: 0.8100
Epoch 12/50
34/34 [==============================] - 0s 2ms/step - loss: 0.4326 - accuracy: 0.9167 - val_l
oss: 0.5195 - val_accuracy: 0.8200
Epoch 13/50
34/34 [==============================] - 0s 2ms/step - loss: 0.3842 - accuracy: 0.9296 - val_l
oss: 0.4949 - val_accuracy: 0.8283
Epoch 14/50
34/34 [==============================] - 0s 2ms/step - loss: 0.3394 - accuracy: 0.9458 - val_l
oss: 0.4867 - val_accuracy: 0.8117
Epoch 15/50
34/34 [==============================] - 0s 2ms/step - loss: 0.3033 - accuracy: 0.9504 - val_l
oss: 0.4578 - val_accuracy: 0.8333
Epoch 16/50
34/34 [==============================] - 0s 2ms/step - loss: 0.2670 - accuracy: 0.9608 - val_l
oss: 0.4410 - val_accuracy: 0.8417
Epoch 17/50
34/34 [==============================] - 0s 2ms/step - loss: 0.2369 - accuracy: 0.9654 - val_l
oss: 0.4294 - val_accuracy: 0.8417
Epoch 18/50
34/34 [==============================] - 0s 2ms/step - loss: 0.2100 - accuracy: 0.9700 - val_l
oss: 0.4212 - val_accuracy: 0.8350
Epoch 19/50
34/34 [==============================] - 0s 2ms/step - loss: 0.1895 - accuracy: 0.9712 - val_l
oss: 0.4132 - val_accuracy: 0.8367
Epoch 20/50
34/34 [==============================] - 0s 2ms/step - loss: 0.1674 - accuracy: 0.9762 - val_l
oss: 0.4102 - val_accuracy: 0.8383
Epoch 21/50
34/34 [==============================] - 0s 2ms/step - loss: 0.1488 - accuracy: 0.9787 - val_l
oss: 0.4068 - val_accuracy: 0.8400
Epoch 22/50
34/34 [==============================] - 0s 2ms/step - loss: 0.1354 - accuracy: 0.9796 - val_l
oss: 0.4073 - val_accuracy: 0.8400
Epoch 23/50
34/34 [==============================] - 0s 2ms/step - loss: 0.1250 - accuracy: 0.9796 - val_l
oss: 0.4081 - val_accuracy: 0.8383
```

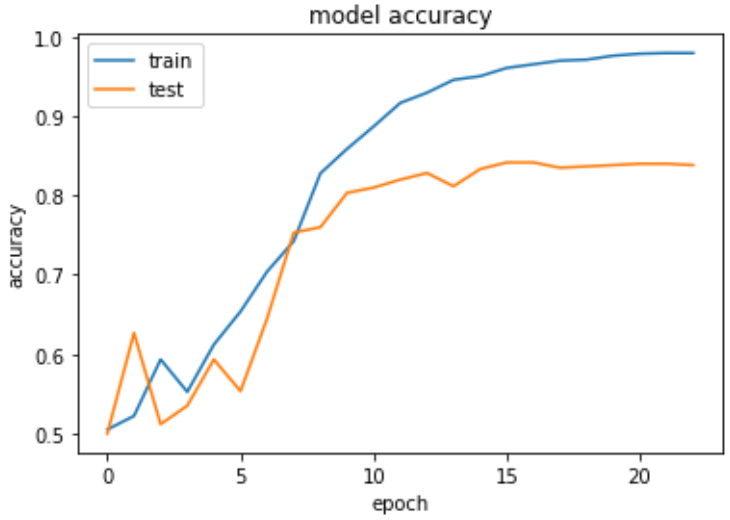
In [17]:

```
model.save('sentimentanalysismodel.h5')
```

In [18]:

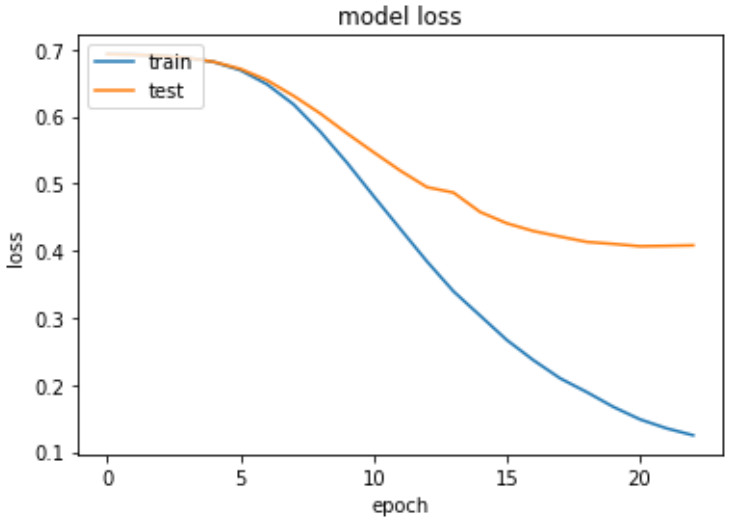
```
# summarize history for accuracy
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

```
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [19]:

```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [ ]: