1. IMPORTS

In [25]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import nltk
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing. text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import RMSprop , Adam
from keras.models import Sequential
from keras.layers import Dense, Embedding, SimpleRNN, LSTM, Bidirectional, Dropou
from nltk.corpus import stopwords
nltk.download('stopwords')
",".join(stopwords.words('english'))
STOPWORDS = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

2. PREPROCESSING THE TEXT

In [3]:
```
df = pd.read_csv('glove-lab-dataset.csv')
df.tail()
```

Out[3]:

	category	text
904	business	marsh executive in guilty plea an executive at...
905	entertainment	bets off after big brother leak a bookmaker ...
906	sport	republic to face china and italy the republic ...
907	politics	butler launches attack on blair former civil s...
908	tech	british library gets wireless net visitors to ...

In [4]:
```
df.shape
```

Out[4]: (909, 2)

In [5]:
```
y =df['category']
X=[]
for review in df['text']:
    filtered_sentence = [w.lower() for w in review.split() if not w in STOPWORDS
    X.append(filtered_sentence)
X = pd.Series(X)
```

In [6]:
```
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(y)
y_seq = np.array(y_tokenizer.texts_to_sequences (y))
```

In [7]:
```
X_token = Tokenizer(num_words=5000,oov_token='<oov>')
X_token.fit_on_texts(X)
word_index = X_token.word_index
X_sequence = X_token.texts_to_sequences (X)
dict(list(word_index.items())[0:15])
```

Out[7]:
```
{'<oov>': 1,
 'said': 2,
 '-': 3,
 'mr': 4,
 'would': 5,
 'also': 6,
 'new': 7,
 'people': 8,
 'us': 9,
 'one': 10,
 'said.': 11,
 'could': 12,
 'last': 13,
 'year': 14,
 'first': 15}
```

In [8]:
```
X_padding= pad_sequences (X_sequence, maxlen=200, padding='post')
```

In [9]:
```
print(y_seq.shape)
print(X_padding.shape)
```

```
(909, 1)
(909, 200)
```

3. DATASET PREPERATION

In [10]:
```
x_train, x_test,y_train,y_test = train_test_split(X_padding, y_seq, train_size=0.
```

In [11]:
```
print(x_train.shape, x_test.shape)
print(y_train.shape, y_test.shape)
```

```
(636, 200) (273, 200)
(636, 1) (273, 1)
```

4. MODEL CREATION

In [12]:
```
#LSTM Model Creation
vocab_size = 5000
embedding_dim = 64
max_length = 200
```

In [13]:
```
model1 = Sequential()
model1.add(Embedding(vocab_size, embedding_dim))
model1.add(LSTM(embedding_dim))
model1.add(Dense(embedding_dim, activation='tanh'))
model1.add(Dense(6, activation='softmax'))
```

In [14]:
```
model1. summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, None, 64)          320000    
                                                                 
 lstm (LSTM)                 (None, 64)                33024     
                                                                 
 dense (Dense)               (None, 64)                4160      
                                                                 
 dense_1 (Dense)             (None, 6)                 390       
                                                                 
=================================================================
Total params: 357,574
Trainable params: 357,574
Non-trainable params: 0
_________________________________________________________________
```

In [15]:
```
model1.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=[
```

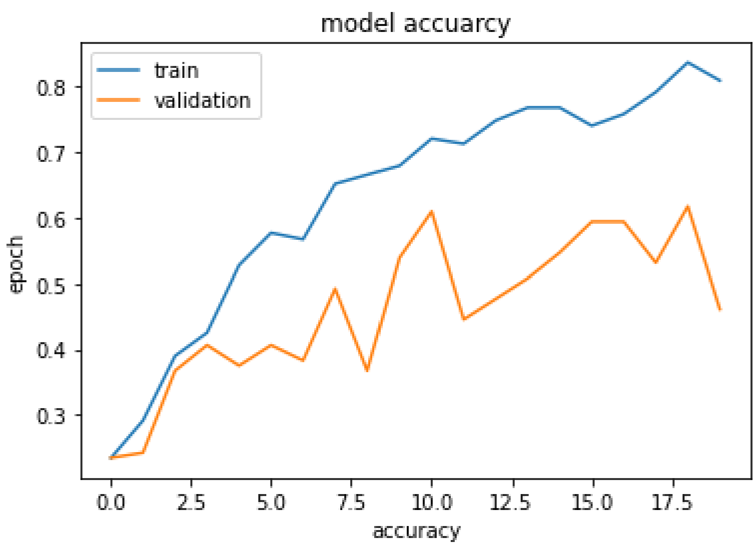
In [16]:
```
history1 = model1.fit(x_train,y_train, epochs=20, verbose=2, validation_split=0.
```

```
Epoch 1/20
16/16 - 4s - loss: 1.7261 - accuracy: 0.2343 - val_loss: 1.6485 - val_accuracy:
0.2344 - 4s/epoch - 236ms/step
Epoch 2/20
16/16 - 1s - loss: 1.5906 - accuracy: 0.2913 - val_loss: 1.5750 - val_accuracy:
0.2422 - 1s/epoch - 88ms/step
Epoch 3/20
16/16 - 1s - loss: 1.5000 - accuracy: 0.3898 - val_loss: 1.5079 - val_accuracy:
0.3672 - 1s/epoch - 87ms/step
Epoch 4/20
16/16 - 1s - loss: 1.3293 - accuracy: 0.4252 - val_loss: 1.4839 - val_accuracy:
0.4062 - 1s/epoch - 90ms/step
Epoch 5/20
16/16 - 1s - loss: 1.2097 - accuracy: 0.5276 - val_loss: 1.4761 - val_accuracy:
0.3750 - 1s/epoch - 90ms/step
Epoch 6/20
16/16 - 1s - loss: 1.1424 - accuracy: 0.5768 - val_loss: 1.3110 - val_accuracy:
0.4062 - 1s/epoch - 91ms/step
Epoch 7/20
16/16 - 1s - loss: 1.0474 - accuracy: 0.5669 - val_loss: 1.3970 - val_accuracy:
0.3828 - 1s/epoch - 91ms/step
Epoch 8/20
16/16 - 2s - loss: 0.9323 - accuracy: 0.6516 - val_loss: 1.2386 - val_accuracy:
0.4922 - 2s/epoch - 110ms/step
Epoch 9/20
16/16 - 2s - loss: 0.8153 - accuracy: 0.6654 - val_loss: 1.4885 - val_accuracy:
0.3672 - 2s/epoch - 103ms/step
Epoch 10/20
16/16 - 1s - loss: 0.7872 - accuracy: 0.6791 - val_loss: 1.1319 - val_accuracy:
0.5391 - 1s/epoch - 90ms/step
Epoch 11/20
16/16 - 1s - loss: 0.6707 - accuracy: 0.7205 - val_loss: 1.0452 - val_accuracy:
0.6094 - 1s/epoch - 90ms/step
Epoch 12/20
16/16 - 1s - loss: 0.6272 - accuracy: 0.7126 - val_loss: 1.4718 - val_accuracy:
0.4453 - 1s/epoch - 91ms/step
Epoch 13/20
16/16 - 1s - loss: 0.5709 - accuracy: 0.7480 - val_loss: 1.2368 - val_accuracy:
0.4766 - 1s/epoch - 90ms/step
Epoch 14/20
16/16 - 1s - loss: 0.5349 - accuracy: 0.7677 - val_loss: 1.3247 - val_accuracy:
0.5078 - 1s/epoch - 89ms/step
Epoch 15/20
16/16 - 1s - loss: 0.5136 - accuracy: 0.7677 - val_loss: 1.3077 - val_accuracy:
0.5469 - 1s/epoch - 91ms/step
Epoch 16/20
16/16 - 1s - loss: 0.5895 - accuracy: 0.7402 - val_loss: 1.0630 - val_accuracy:
0.5938 - 1s/epoch - 88ms/step
Epoch 17/20
16/16 - 1s - loss: 0.5659 - accuracy: 0.7579 - val_loss: 1.0076 - val_accuracy:
0.5938 - 1s/epoch - 90ms/step
Epoch 18/20
16/16 - 2s - loss: 0.4687 - accuracy: 0.7913 - val_loss: 1.2073 - val_accuracy:
0.5312 - 2s/epoch - 95ms/step
```

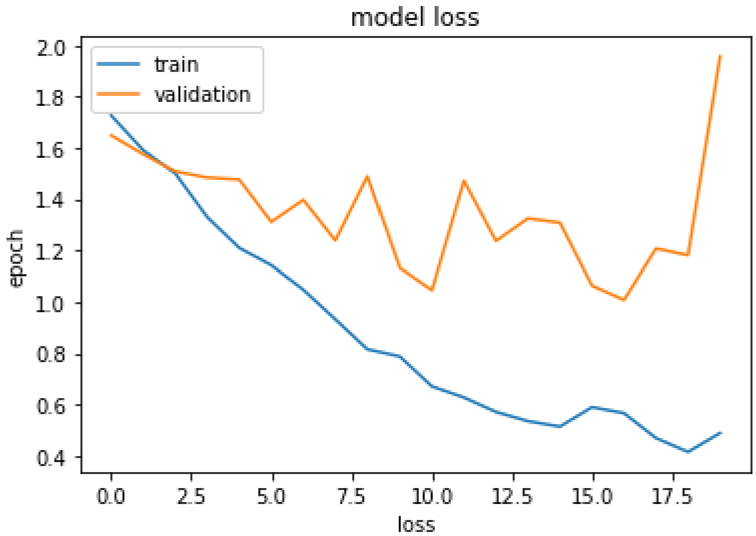
```
Epoch 19/20
16/16 - 1s - loss: 0.4148 - accuracy: 0.8366 - val_loss: 1.1812 - val_accuracy:
0.6172 - 1s/epoch - 91ms/step
Epoch 20/20
16/16 - 1s - loss: 0.4884 - accuracy: 0.8091 - val_loss: 1.9568 - val_accuracy:
0.4609 - 1s/epoch - 91ms/step
```

In [17]:

```
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title('model accuarcy')
plt.xlabel('accuracy')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



In [18]:
```
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('model loss')
plt.xlabel('loss')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



In [19]:
```
print("loss: ", model1.evaluate(x_test,y_test, verbose=0)[0])
print("accuarcy: ", model1.evaluate(x_test, y_test, verbose=0)[1])
```

```
loss:  1.8595274686813354
accuarcy:  0.5018315315246582
```

6. VARIATIONS

In [20]:
```
#CNN-LSTM Model Creation
```

In [26]:
```
model2 = Sequential()
model2.add(Embedding(vocab_size, embedding_dim))
model2.add(Conv1D(filters=32, kernel_size=5, strides=1, activation='relu'))
model2.add(MaxPooling1D((2)))
model2.add(LSTM(embedding_dim))
model2.add(Dense(128, activation= 'relu'))
model2.add(Dense(6,activation='softmax'))
```

In [27]:
```
model2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, None, 64)          320000    
                                                                 
 conv1d_1 (Conv1D)           (None, None, 32)          10272     
                                                                 
 max_pooling1d (MaxPooling1D  (None, None, 32)         0         
 )                                                               
                                                                 
 lstm_1 (LSTM)               (None, 64)                24832     
                                                                 
 dense_2 (Dense)             (None, 128)               8320      
                                                                 
 dense_3 (Dense)             (None, 6)                 774       
                                                                 
=================================================================
Total params: 364,198
Trainable params: 364,198
Non-trainable params: 0
_________________________________________________________________
```

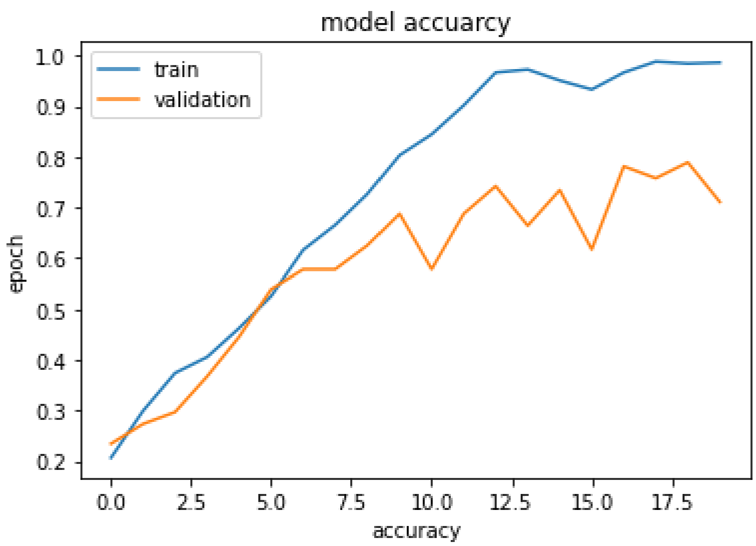
In [28]:
```
model2.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=[
```

In [29]:
```
history2 = model2. fit(x_train,y_train, epochs=20,validation_split=0.2, verbose=
```

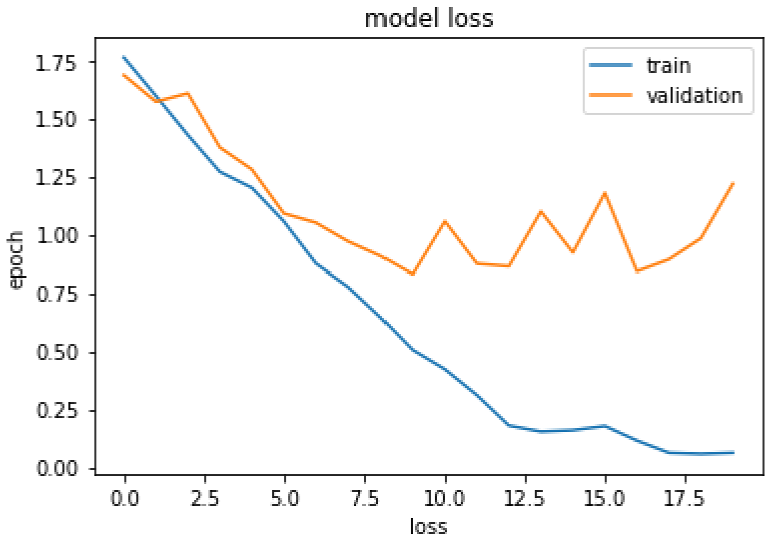
```
Epoch 1/20
16/16 - 3s - loss: 1.7624 - accuracy: 0.2067 - val_loss: 1.6859 - val_accuracy:
0.2344 - 3s/epoch - 187ms/step
Epoch 2/20
16/16 - 1s - loss: 1.5971 - accuracy: 0.2992 - val_loss: 1.5716 - val_accuracy:
0.2734 - 906ms/epoch - 57ms/step
Epoch 3/20
16/16 - 1s - loss: 1.4279 - accuracy: 0.3740 - val_loss: 1.6075 - val_accuracy:
0.2969 - 964ms/epoch - 60ms/step
Epoch 4/20
16/16 - 1s - loss: 1.2694 - accuracy: 0.4055 - val_loss: 1.3745 - val_accuracy:
0.3672 - 906ms/epoch - 57ms/step
Epoch 5/20
16/16 - 1s - loss: 1.2007 - accuracy: 0.4626 - val_loss: 1.2797 - val_accuracy:
0.4453 - 970ms/epoch - 61ms/step
Epoch 6/20
16/16 - 1s - loss: 1.0557 - accuracy: 0.5256 - val_loss: 1.0900 - val_accuracy:
0.5391 - 921ms/epoch - 58ms/step
Epoch 7/20
16/16 - 1s - loss: 0.8756 - accuracy: 0.6161 - val_loss: 1.0503 - val_accuracy:
0.5781 - 942ms/epoch - 59ms/step
Epoch 8/20
16/16 - 1s - loss: 0.7759 - accuracy: 0.6654 - val_loss: 0.9704 - val_accuracy:
0.5781 - 933ms/epoch - 58ms/step
Epoch 9/20
16/16 - 1s - loss: 0.6462 - accuracy: 0.7264 - val_loss: 0.9084 - val_accuracy:
0.6250 - 910ms/epoch - 57ms/step
Epoch 10/20
16/16 - 1s - loss: 0.5047 - accuracy: 0.8031 - val_loss: 0.8297 - val_accuracy:
0.6875 - 935ms/epoch - 58ms/step
Epoch 11/20
16/16 - 1s - loss: 0.4218 - accuracy: 0.8445 - val_loss: 1.0563 - val_accuracy:
0.5781 - 970ms/epoch - 61ms/step
Epoch 12/20
16/16 - 1s - loss: 0.3102 - accuracy: 0.9016 - val_loss: 0.8746 - val_accuracy:
0.6875 - 932ms/epoch - 58ms/step
Epoch 13/20
16/16 - 1s - loss: 0.1790 - accuracy: 0.9665 - val_loss: 0.8632 - val_accuracy:
0.7422 - 961ms/epoch - 60ms/step
Epoch 14/20
16/16 - 1s - loss: 0.1530 - accuracy: 0.9724 - val_loss: 1.0986 - val_accuracy:
0.6641 - 948ms/epoch - 59ms/step
Epoch 15/20
16/16 - 1s - loss: 0.1591 - accuracy: 0.9508 - val_loss: 0.9220 - val_accuracy:
0.7344 - 967ms/epoch - 60ms/step
Epoch 16/20
16/16 - 1s - loss: 0.1770 - accuracy: 0.9331 - val_loss: 1.1783 - val_accuracy:
0.6172 - 946ms/epoch - 59ms/step
Epoch 17/20
16/16 - 1s - loss: 0.1144 - accuracy: 0.9665 - val_loss: 0.8418 - val_accuracy:
0.7812 - 935ms/epoch - 58ms/step
Epoch 18/20
16/16 - 1s - loss: 0.0616 - accuracy: 0.9882 - val_loss: 0.8927 - val_accuracy:
0.7578 - 937ms/epoch - 59ms/step
```

```
Epoch 19/20
16/16 - 1s - loss: 0.0570 - accuracy: 0.9843 - val_loss: 0.9830 - val_accuracy:
0.7891 - 948ms/epoch - 59ms/step
Epoch 20/20
16/16 - 1s - loss: 0.0616 - accuracy: 0.9862 - val_loss: 1.2179 - val_accuracy:
0.7109 - 912ms/epoch - 57ms/step
```

In [30]:
```
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('model accuarcy')
plt.xlabel('accuracy')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



In [31]:
```
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model loss')
plt.xlabel('loss')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



yes,performance of model1 (CNN-LSTM layer) is much better than model (LSTM layer) acuuracy have been improved a lot

In [32]:
```
#Pre-trained Model Creation
! wget --no-check-certificate \
    http://nlp.stanford.edu/data/glove.6B.zip \
    -O /tmp/glove.6B.zip
```

```
--2022-10-14 04:21:53--  http://nlp.stanford.edu/data/glove.6B.zip (http://nlp.stanford.edu/data/glove.6B.zip)
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip (https://nlp.stanford.edu/data/glove.6B.zip) [following]
--2022-10-14 04:21:54--  https://nlp.stanford.edu/data/glove.6B.zip (https://nlp.stanford.edu/data/glove.6B.zip)
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip (https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip) [following]
--2022-10-14 04:21:54--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip (https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip)
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘/tmp/glove.6B.zip’

/tmp/glove.6B.zip   100%[===================>] 822.24M  5.09MB/s    in 2m 41s

2022-10-14 04:24:36 (5.11 MB/s) - ‘/tmp/glove.6B.zip’ saved [862182613/862182613]
```

7. TYPES OF RNN LAYERS

In [34]:
```
import os, zipfile
with zipfile.ZipFile('/tmp/glove.6B.zip', 'r') as zip_rf:
    zip_rf.extractall('/tmp/glove')
```

In [36]:
```
embeddings_index = {}
f = open('/tmp/glove/glove.6B.100d.txt')
for line in f:
  values = line.split()
  word = values[0]
  coefs = np.asarray(values[1:], dtype='float32')
  embeddings_index[word] = coefs

f.close()
```

In [37]:
```
print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 400000 word vectors.
```

In [38]:
```
 embedding_matrix = np.zeros((len(word_index) + 1, 100))
for word, i in word_index.items():
  embedding_vector = embeddings_index.get(word)
if embedding_vector is not None:
# words not found in embedding index will be all-zeros. embedding_matrix[i] = emb
  embedding_matrix[i] = embedding_vector
```

In [43]:
```

embedding_layer = Embedding(input_dim=len(word_index) + 1, output_dim=100)
```

In [46]:
```
model3 = Sequential()
model3.add(embedding_layer)
model3.add(Conv1D(filters=64, kernel_size=5, strides=1, activation='relu'))
model3.add(MaxPooling1D((2)))
model3.add(LSTM(100))
model3.add(Dense(64, activation= 'relu'))
model3.add(Dense(6, activation='softmax'))
model3. summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, None, 100)         2805100

 conv1d_2 (Conv1D)           (None, None, 64)          32064

 max_pooling1d_1 (MaxPooling  (None, None, 64)         0
 1D)

 lstm_2 (LSTM)               (None, 100)               66000

 dense_4 (Dense)             (None, 64)                6464

 dense_5 (Dense)             (None, 6)                 390

=================================================================
Total params: 2,910,018
Trainable params: 2,910,018
Non-trainable params: 0
_________________________________________________________________
```

In [50]:
```
model3.compile(optimizer=RMSprop(0.0001), loss='sparse_categorical_crossentropy',
```

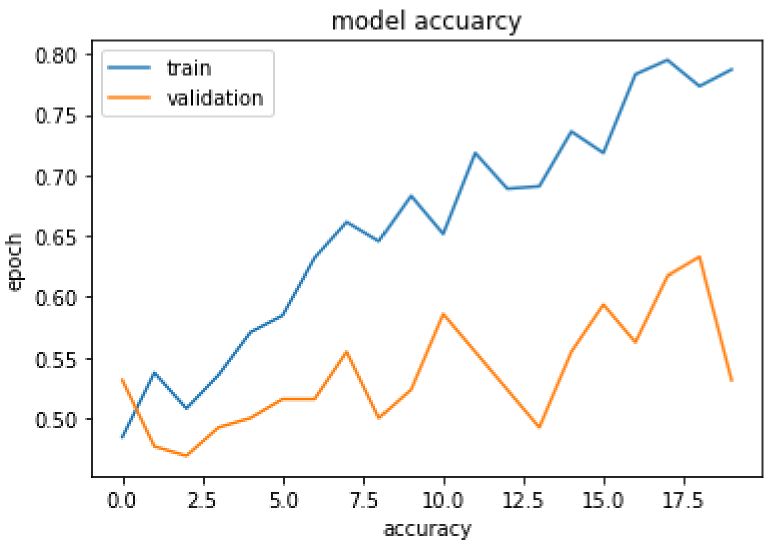
In [51]:
```
history3 = model3. fit(x_train,y_train, epochs=20, validation_split=0.2, verbose=
```

```
Epoch 1/20
16/16 - 4s - loss: 1.1890 - accuracy: 0.4843 - val_loss: 1.1542 - val_accuracy:
0.5312 - 4s/epoch - 250ms/step
Epoch 2/20
16/16 - 2s - loss: 1.1364 - accuracy: 0.5374 - val_loss: 1.1693 - val_accuracy:
0.4766 - 2s/epoch - 106ms/step
Epoch 3/20
16/16 - 2s - loss: 1.1441 - accuracy: 0.5079 - val_loss: 1.1530 - val_accuracy:
0.4688 - 2s/epoch - 105ms/step
Epoch 4/20
16/16 - 2s - loss: 1.1007 - accuracy: 0.5354 - val_loss: 1.1687 - val_accuracy:
0.4922 - 2s/epoch - 108ms/step
Epoch 5/20
16/16 - 2s - loss: 1.0815 - accuracy: 0.5709 - val_loss: 1.2042 - val_accuracy:
0.5000 - 2s/epoch - 106ms/step
Epoch 6/20
16/16 - 2s - loss: 1.0384 - accuracy: 0.5846 - val_loss: 1.0964 - val_accuracy:
0.5156 - 2s/epoch - 104ms/step
Epoch 7/20
16/16 - 2s - loss: 0.9830 - accuracy: 0.6319 - val_loss: 1.0748 - val_accuracy:
0.5156 - 2s/epoch - 105ms/step
Epoch 8/20
16/16 - 2s - loss: 0.9422 - accuracy: 0.6614 - val_loss: 1.0837 - val_accuracy:
0.5547 - 2s/epoch - 105ms/step
Epoch 9/20
16/16 - 2s - loss: 0.9600 - accuracy: 0.6457 - val_loss: 1.0795 - val_accuracy:
0.5000 - 2s/epoch - 107ms/step
Epoch 10/20
16/16 - 2s - loss: 0.8579 - accuracy: 0.6831 - val_loss: 1.0682 - val_accuracy:
0.5234 - 2s/epoch - 107ms/step
Epoch 11/20
16/16 - 2s - loss: 0.8763 - accuracy: 0.6516 - val_loss: 1.0053 - val_accuracy:
0.5859 - 2s/epoch - 106ms/step
Epoch 12/20
16/16 - 2s - loss: 0.7800 - accuracy: 0.7185 - val_loss: 1.0609 - val_accuracy:
0.5547 - 2s/epoch - 106ms/step
Epoch 13/20
16/16 - 2s - loss: 0.7923 - accuracy: 0.6890 - val_loss: 1.1097 - val_accuracy:
0.5234 - 2s/epoch - 109ms/step
Epoch 14/20
16/16 - 2s - loss: 0.7827 - accuracy: 0.6909 - val_loss: 1.1385 - val_accuracy:
0.4922 - 2s/epoch - 106ms/step
Epoch 15/20
16/16 - 2s - loss: 0.7384 - accuracy: 0.7362 - val_loss: 0.9499 - val_accuracy:
0.5547 - 2s/epoch - 105ms/step
Epoch 16/20
16/16 - 2s - loss: 0.7409 - accuracy: 0.7185 - val_loss: 0.9521 - val_accuracy:
0.5938 - 2s/epoch - 106ms/step
Epoch 17/20
16/16 - 2s - loss: 0.6478 - accuracy: 0.7835 - val_loss: 1.0002 - val_accuracy:
0.5625 - 2s/epoch - 131ms/step
Epoch 18/20
16/16 - 3s - loss: 0.6233 - accuracy: 0.7953 - val_loss: 0.9831 - val_accuracy:
0.6172 - 3s/epoch - 157ms/step
Epoch 19/20
```

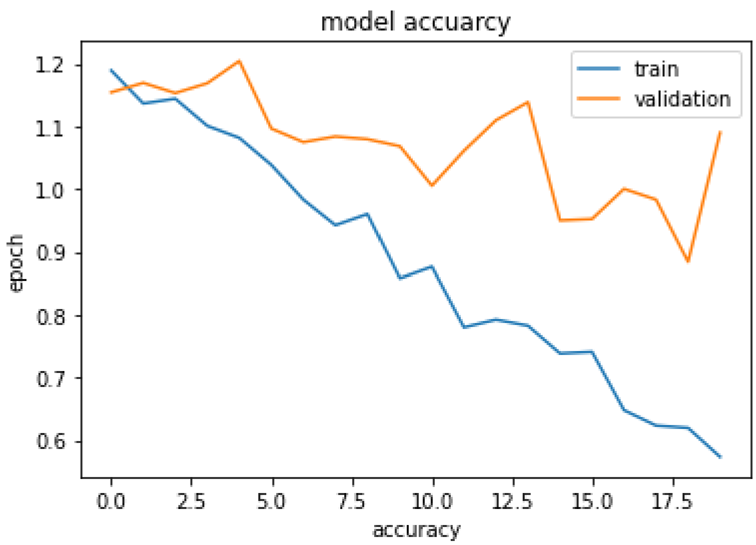
```
16/16 - 2s - loss: 0.6196 - accuracy: 0.7736 - val_loss: 0.8839 - val_accuracy:
0.6328 - 2s/epoch - 107ms/step
Epoch 20/20
16/16 - 2s - loss: 0.5734 - accuracy: 0.7874 - val_loss: 1.0898 - val_accuracy:
0.5312 - 2s/epoch - 106ms/step
```

In [52]:

```
plt.plot(history3.history['accuracy'])
plt.plot(history3.history['val_accuracy'])
plt.title('model accuarcy')
plt.xlabel('accuracy')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



In [53]:
```
plt.plot(history3.history['loss'])
plt.plot(history3.history['val_loss'])
plt.title('model accuarcy')
plt.xlabel('accuracy')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



In [55]:
```
score = model3.evaluate(x_test,y_test,verbose=0)
print("loss: ", score[0])
print("accuracy: ", score[1])
```

```
loss:  1.044947862625122
accuracy:  0.5311355590820312
```

Improvements Try dropouts and see if you can improve the performance of your models.

In [56]:
```
model4 = Sequential()
model4.add(Embedding(vocab_size, embedding_dim))
model4.add(Conv1D(filters=32, kernel_size=3, strides=1, activation='tanh'))
model4.add(MaxPooling1D((2)))
model4.add(LSTM(embedding_dim))
model4.add(Dropout(0.3))
model4.add(Dense(512, activation='relu'))
model4.add(Dense(6, activation='softmax'))
model4. summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_5 (Embedding)     (None, None, 64)          320000    
                                                                 
 conv1d_3 (Conv1D)           (None, None, 32)          6176      
                                                                 
 max_pooling1d_2 (MaxPooling  (None, None, 32)         0         
 1D)                                                             
                                                                 
 lstm_3 (LSTM)               (None, 64)                24832     
                                                                 
 dropout (Dropout)           (None, 64)                0         
                                                                 
 dense_6 (Dense)             (None, 512)               33280     
                                                                 
 dense_7 (Dense)             (None, 6)                 3078      
                                                                 
=================================================================
Total params: 387,366
Trainable params: 387,366
Non-trainable params: 0
_________________________________________________________________
```

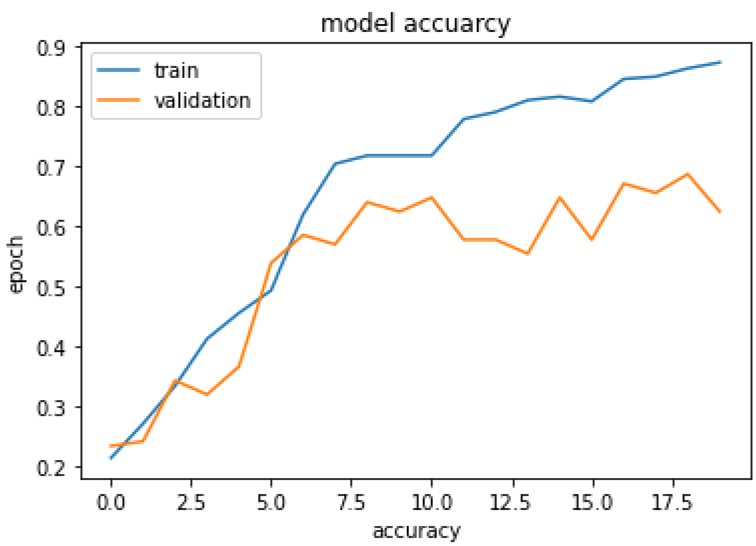
In [57]:
```
model4.compile(optimizer='adam',loss='sparse_categorical_crossentropy', metrics=[
```

In [58]:
```
history4 = model4.fit(x_train,y_train, epochs=20,validation_split=0.2, verbose=2)
```

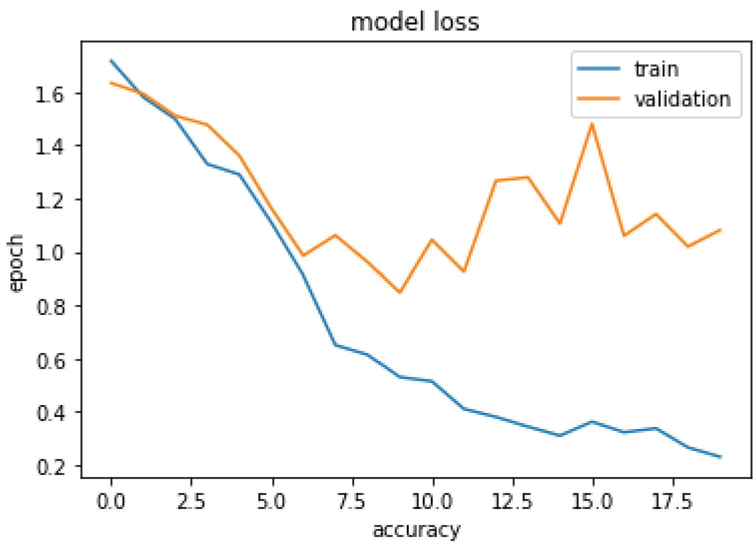
```
Epoch 1/20
16/16 - 3s - loss: 1.7163 - accuracy: 0.2146 - val_loss: 1.6333 - val_accuracy:
0.2344 - 3s/epoch - 194ms/step
Epoch 2/20
16/16 - 1s - loss: 1.5804 - accuracy: 0.2717 - val_loss: 1.5930 - val_accuracy:
0.2422 - 949ms/epoch - 59ms/step
Epoch 3/20
16/16 - 1s - loss: 1.4980 - accuracy: 0.3346 - val_loss: 1.5102 - val_accuracy:
0.3438 - 933ms/epoch - 58ms/step
Epoch 4/20
16/16 - 1s - loss: 1.3289 - accuracy: 0.4134 - val_loss: 1.4767 - val_accuracy:
0.3203 - 972ms/epoch - 61ms/step
Epoch 5/20
16/16 - 1s - loss: 1.2898 - accuracy: 0.4567 - val_loss: 1.3609 - val_accuracy:
0.3672 - 948ms/epoch - 59ms/step
Epoch 6/20
16/16 - 1s - loss: 1.1095 - accuracy: 0.4941 - val_loss: 1.1633 - val_accuracy:
0.5391 - 959ms/epoch - 60ms/step
Epoch 7/20
16/16 - 1s - loss: 0.9115 - accuracy: 0.6201 - val_loss: 0.9850 - val_accuracy:
0.5859 - 971ms/epoch - 61ms/step
Epoch 8/20
16/16 - 1s - loss: 0.6506 - accuracy: 0.7047 - val_loss: 1.0613 - val_accuracy:
0.5703 - 975ms/epoch - 61ms/step
Epoch 9/20
16/16 - 1s - loss: 0.6144 - accuracy: 0.7185 - val_loss: 0.9612 - val_accuracy:
0.6406 - 918ms/epoch - 57ms/step
Epoch 10/20
16/16 - 1s - loss: 0.5302 - accuracy: 0.7185 - val_loss: 0.8477 - val_accuracy:
0.6250 - 922ms/epoch - 58ms/step
Epoch 11/20
16/16 - 1s - loss: 0.5145 - accuracy: 0.7185 - val_loss: 1.0444 - val_accuracy:
0.6484 - 1s/epoch - 63ms/step
Epoch 12/20
16/16 - 2s - loss: 0.4106 - accuracy: 0.7795 - val_loss: 0.9246 - val_accuracy:
0.5781 - 2s/epoch - 111ms/step
Epoch 13/20
16/16 - 2s - loss: 0.3805 - accuracy: 0.7913 - val_loss: 1.2662 - val_accuracy:
0.5781 - 2s/epoch - 104ms/step
Epoch 14/20
16/16 - 2s - loss: 0.3439 - accuracy: 0.8110 - val_loss: 1.2792 - val_accuracy:
0.5547 - 2s/epoch - 123ms/step
Epoch 15/20
16/16 - 1s - loss: 0.3102 - accuracy: 0.8169 - val_loss: 1.1052 - val_accuracy:
0.6484 - 1s/epoch - 85ms/step
Epoch 16/20
16/16 - 2s - loss: 0.3626 - accuracy: 0.8091 - val_loss: 1.4796 - val_accuracy:
0.5781 - 2s/epoch - 108ms/step
Epoch 17/20
16/16 - 1s - loss: 0.3228 - accuracy: 0.8465 - val_loss: 1.0594 - val_accuracy:
0.6719 - 984ms/epoch - 62ms/step
Epoch 18/20
16/16 - 2s - loss: 0.3375 - accuracy: 0.8504 - val_loss: 1.1409 - val_accuracy:
0.6562 - 2s/epoch - 111ms/step
```

```
Epoch 19/20
16/16 - 1s - loss: 0.2657 - accuracy: 0.8642 - val_loss: 1.0189 - val_accuracy:
0.6875 - 977ms/epoch - 61ms/step
Epoch 20/20
16/16 - 1s - loss: 0.2305 - accuracy: 0.8740 - val_loss: 1.0807 - val_accuracy:
0.6250 - 935ms/epoch - 58ms/step
```

In [59]:
```
plt.plot(history4.history['accuracy'])
plt.plot(history4.history['val_accuracy'])
plt.title('model accuarcy')
plt.xlabel('accuracy')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



In [60]:
```
plt.plot(history4.history['loss'])
plt.plot(history4.history['val_loss'])
plt.title('model loss')
plt.xlabel('accuracy')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



In [63]:
```
score = model4.evaluate(x_test,y_test,verbose=0)
print("loss: ", score[0])
print("accuracy: ", score[1])
```

```
loss:  0.7557295560836792
accuracy:  0.6556776762008667
```

Split your dataset with 20% testing and observe your performance.

In [64]:
```
x_train,x_test,y_train,y_test = train_test_split(X_padding,y_seq, train_size=0.8)
```

In [65]:
```
print(x_train.shape, x_test.shape)
print(y_train.shape, y_test.shape)
```

```
(727, 200) (182, 200)
(727, 1) (182, 1)
```

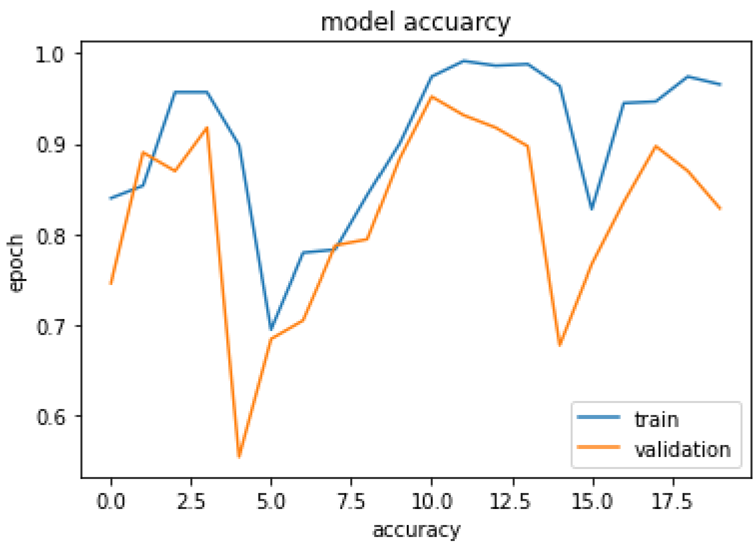
In [66]:
```
history5 = model2. fit(x_train,y_train, epochs=20,validation_split=0.2, verbose=2
```

```
Epoch 1/20
19/19 - 2s - loss: 0.5414 - accuracy: 0.8399 - val_loss: 0.7343 - val_accurac
y: 0.7466 - 2s/epoch - 96ms/step
Epoch 2/20
19/19 - 1s - loss: 0.3809 - accuracy: 0.8537 - val_loss: 0.3367 - val_accurac
y: 0.8904 - 1s/epoch - 61ms/step
Epoch 3/20
19/19 - 1s - loss: 0.1774 - accuracy: 0.9570 - val_loss: 0.4025 - val_accurac
y: 0.8699 - 1s/epoch - 62ms/step
Epoch 4/20
19/19 - 1s - loss: 0.1467 - accuracy: 0.9570 - val_loss: 0.3145 - val_accurac
y: 0.9178 - 1s/epoch - 63ms/step
Epoch 5/20
19/19 - 1s - loss: 0.4361 - accuracy: 0.8985 - val_loss: 1.4710 - val_accurac
y: 0.5548 - 1s/epoch - 63ms/step
Epoch 6/20
19/19 - 1s - loss: 0.8430 - accuracy: 0.6954 - val_loss: 0.7913 - val_accurac
y: 0.6849 - 1s/epoch - 61ms/step
Epoch 7/20
19/19 - 1s - loss: 0.4682 - accuracy: 0.7797 - val_loss: 0.6805 - val_accurac
y: 0.7055 - 1s/epoch - 62ms/step
Epoch 8/20
19/19 - 1s - loss: 0.4255 - accuracy: 0.7831 - val_loss: 0.5612 - val_accurac
y: 0.7877 - 1s/epoch - 62ms/step
Epoch 9/20
19/19 - 1s - loss: 0.3328 - accuracy: 0.8434 - val_loss: 0.5010 - val_accurac
y: 0.7945 - 1s/epoch - 62ms/step
Epoch 10/20
19/19 - 1s - loss: 0.2903 - accuracy: 0.9002 - val_loss: 0.4433 - val_accurac
y: 0.8836 - 1s/epoch - 61ms/step
Epoch 11/20
19/19 - 1s - loss: 0.1968 - accuracy: 0.9742 - val_loss: 0.3192 - val_accurac
y: 0.9521 - 1s/epoch - 62ms/step
Epoch 12/20
19/19 - 1s - loss: 0.1142 - accuracy: 0.9914 - val_loss: 0.3103 - val_accurac
y: 0.9315 - 1s/epoch - 61ms/step
Epoch 13/20
19/19 - 1s - loss: 0.0885 - accuracy: 0.9862 - val_loss: 0.2814 - val_accurac
y: 0.9178 - 1s/epoch - 62ms/step
Epoch 14/20
19/19 - 1s - loss: 0.0623 - accuracy: 0.9880 - val_loss: 0.3483 - val_accurac
y: 0.8973 - 1s/epoch - 61ms/step
Epoch 15/20
19/19 - 1s - loss: 0.1537 - accuracy: 0.9639 - val_loss: 0.8570 - val_accurac
y: 0.6781 - 1s/epoch - 59ms/step
Epoch 16/20
19/19 - 1s - loss: 0.4225 - accuracy: 0.8279 - val_loss: 0.6458 - val_accurac
y: 0.7671 - 1s/epoch - 61ms/step
Epoch 17/20
19/19 - 1s - loss: 0.2179 - accuracy: 0.9449 - val_loss: 0.4977 - val_accurac
y: 0.8356 - 1s/epoch - 61ms/step
Epoch 18/20
19/19 - 1s - loss: 0.1818 - accuracy: 0.9466 - val_loss: 0.3790 - val_accurac
y: 0.8973 - 1s/epoch - 62ms/step
```

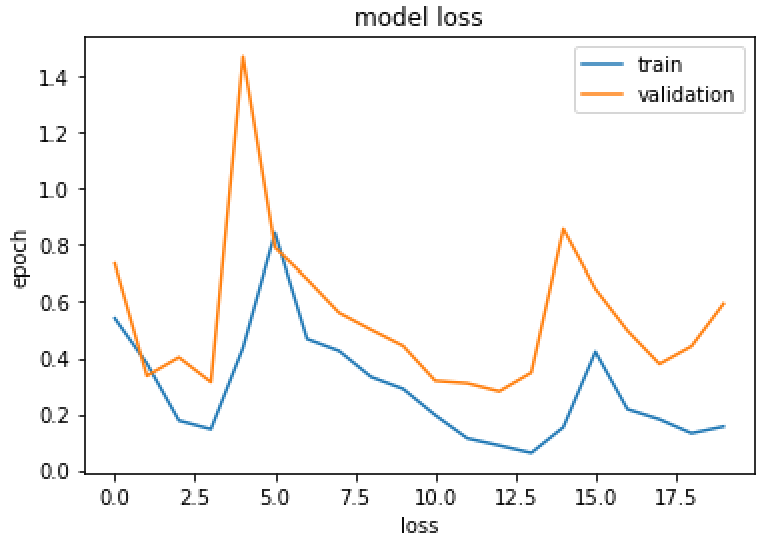
```
Epoch 19/20
19/19 - 1s - loss: 0.1321 - accuracy: 0.9742 - val_loss: 0.4422 - val_accurac
y: 0.8699 - 1s/epoch - 62ms/step
Epoch 20/20
19/19 - 1s - loss: 0.1562 - accuracy: 0.9656 - val_loss: 0.5938 - val_accurac
y: 0.8288 - 1s/epoch - 61ms/step
```

In [68]:

```python
plt.plot(history5.history['accuracy'])
plt.plot(history5.history['val_accuracy'])
plt.title('model accuarcy')
plt.xlabel('accuracy')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



In [69]:
```
plt.plot(history5.history['loss'])
plt.plot(history5.history['val_loss'])
plt.title('model loss')
plt.xlabel('loss')
plt.ylabel('epoch')
plt.legend(['train', 'validation'])
plt.show()
```



In [70]:
```
score = model2.evaluate(x_test,y_test, verbose=0)
print("loss: ", score[0])
print("accuracy: ", score[1])
```

```
loss:  0.660285472869873
accuracy:  0.807692289352417
```

In [71]:
```
 txt = ["Australia claimed the crucial wicket of Sri Lankan opener Pathum Nissank
```

In [72]:
```
tokenizer = Tokenizer(num_words=5000,oov_token='<oov>')
tokenizer.fit_on_texts(txt)
seq = tokenizer.texts_to_sequences (txt)
padded = pad_sequences(seq, maxlen=300)
pred = model1.predict(padded)
```

```
1/1 [==============================] - 1s 566ms/step
```

In [73]:
```
labels = ['tech', 'bussiness', 'politics', 'sport', 'entertainment']
print(pred, labels[np.argmax(pred)])
```

```
[[1.69721004e-02 7.34139408e-04 1.12347655e-01 8.49025324e-03
  8.60715330e-01 7.40452204e-04]] entertainment
```

In [74]:
```
 txt = ["Australia claimed the crucial wicket of Sri Lankan opener Pathum Nissank
```

In [75]:
```
tokenizer = Tokenizer(num_words=5000,oov_token='<oov>')
tokenizer.fit_on_texts(txt)
seq = tokenizer.texts_to_sequences (txt)
padded = pad_sequences (seq, maxlen=300)
pred = model2.predict(padded)
```

```
1/1 [==============================] - 0s 370ms/step
```

In [76]:
```
labels = ['tech', 'bussiness', 'politics', 'sport', 'entertainment']
print(pred, labels[np.argmax(pred)])
```

```
[[1.2370144e-07 9.7057319e-01 4.8245133e-06 1.5209980e-09 2.6251193e-08
  2.9421855e-02]] bussiness
```

In [77]:
```
 txt = ["Australia claimed the crucial wicket of Sri Lankan opener Pathum Nissank
tokenizer = Tokenizer(num_words=5000,oov_token='<oov>')
tokenizer.fit_on_texts(txt)
seq = tokenizer.texts_to_sequences(txt)
padded = pad_sequences (seq, maxlen=300)
pred = model3.predict(padded)
#on model3
labels = ['tech', 'bussiness', 'politics', 'sport', 'entertainment']
print(pred, labels[np.argmax(pred)])
```

```
1/1 [==============================] - 1s 513ms/step
[[0.00119656 0.897877   0.02025985 0.00960327 0.00183333 0.06922997]] bussiness
```

In [ ]:
```
txt = ["Australia claimed the crucial wicket of Sri Lankan opener Pathum Nissanka
tokenizer = Tokenizer(num_words=5000,oov_token='<oov>')
tokenizer.fit_on_texts(txt)
seq = tokenizer.texts_to_sequences(txt)
padded = pad_sequences (seq, maxlen=300)
pred = model3.predict(padded)
#on model3
labels = ['tech', 'bussiness', 'politics', 'sport', 'entertainment']
print(pred, labels[np.argmax(pred)])
```