Classification of Different Tech Feild Using NLP

Here the provided data is about the Technology in which a company is work. As the company Root2AI focus on providing Tech solution to company. This dataset seems to have a lots of potential to provide solution to companies. We could use this classification model for building a chatbot which provides customized introduction message to customer.

For example if a customer need web-services and when they interact with the company this model would help bot to understand what customer needs and then chat bot would text or email the customer with information technology company is working and emphasizing the works company did in web-services

Different Libraries used

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import re
from nltk.stem import WordNetLemmatizer
import nltk
from nltk.corpus import stopwords
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import keras
from keras.layers import Embedding,LSTM,Dense,Input,SimpleRNN,GRU
from keras.callbacks import ReduceLROnPlateau,EarlyStopping,TerminateOnNaN
from keras.models import Sequential
from sklearn.preprocessing import LabelEncoder
from keras.layers import GlobalMaxPooling1D
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedShuffleSplit
import warnings
warnings.filterwarnings('ignore')
```

In [14]:
```
data=pd.read_csv('../input/root2ai/root2ai - Data.csv')
data.head()
```

Out[14]:

	Text	Target
0	reserve bank forming expert committee based in...	Blockchain
1	director could play role financial system	Blockchain
2	preliminary discuss secure transaction study r...	Blockchain
3	security indeed prove essential transforming f...	Blockchain
4	bank settlement normally take three days based...	Blockchain

In [15]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22704 entries, 0 to 22703
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Text    22701 non-null  object
 1   Target  22704 non-null  object
dtypes: object(2)
memory usage: 354.9+ KB
```

Few null values are there in Text column so droping it

In [16]:
```
data.dropna(inplace=True)
```

In [17]:
```
data.Target.value_counts()
```

Out[17]:
```
FinTech              8551
Cyber Security       2640
Bigdata              2267
Reg Tech             2206
credit reporting     1748
Blockchain           1375
Neobanks             1069
Microservices         974
Stock Trading         787
Robo Advising         737
Data Security         347
Name: Target, dtype: int64
```

After a through look at dataset i realized that there is a great problem with dataset.i,e. some of the Texts are just few words and that also a very common word that even the human can't categoried it. So i have selected a threshold of 4 words if texts have less than 4 word we can simply drop them as they can be considered as outliers.

In [18]:
```
def remove_short_text(texts):
    words=texts.split()
    if len(words)<=3:
        return None
    else:
        return texts

data.Text=data.Text.apply(remove_short_text)
```

In [21]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18708 entries, 0 to 22703
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Text    18708 non-null  object
 1   Target  18708 non-null  object
dtypes: object(2)
memory usage: 438.5+ KB
```

In [20]:
```
data.dropna(inplace=True)
```

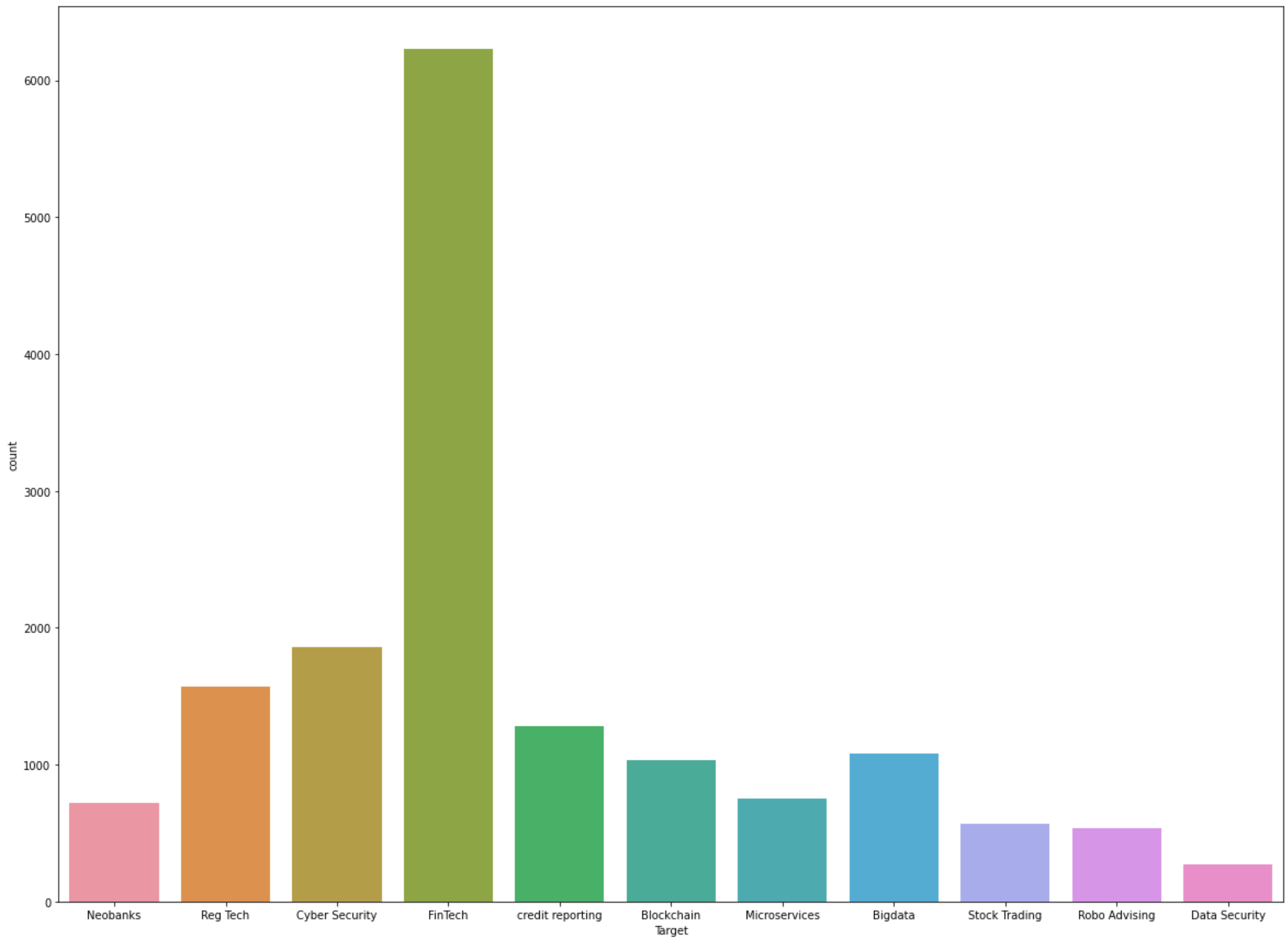
For uniform data-spliting among train and test dataset we use stratifiedShuffleSplit.

In [22]:
```
split=StratifiedShuffleSplit(test_size=0.15,random_state=42)
for train_index,test_index in split.split(data,data['Target']):
    train_data=data.iloc[train_index]
    test_data=data.iloc[test_index]
```

In [23]:
```
train_data.shape
```
Out[23]: (15901, 2)

In [24]:
```
test_data.shape
```
Out[24]: (2807, 2)

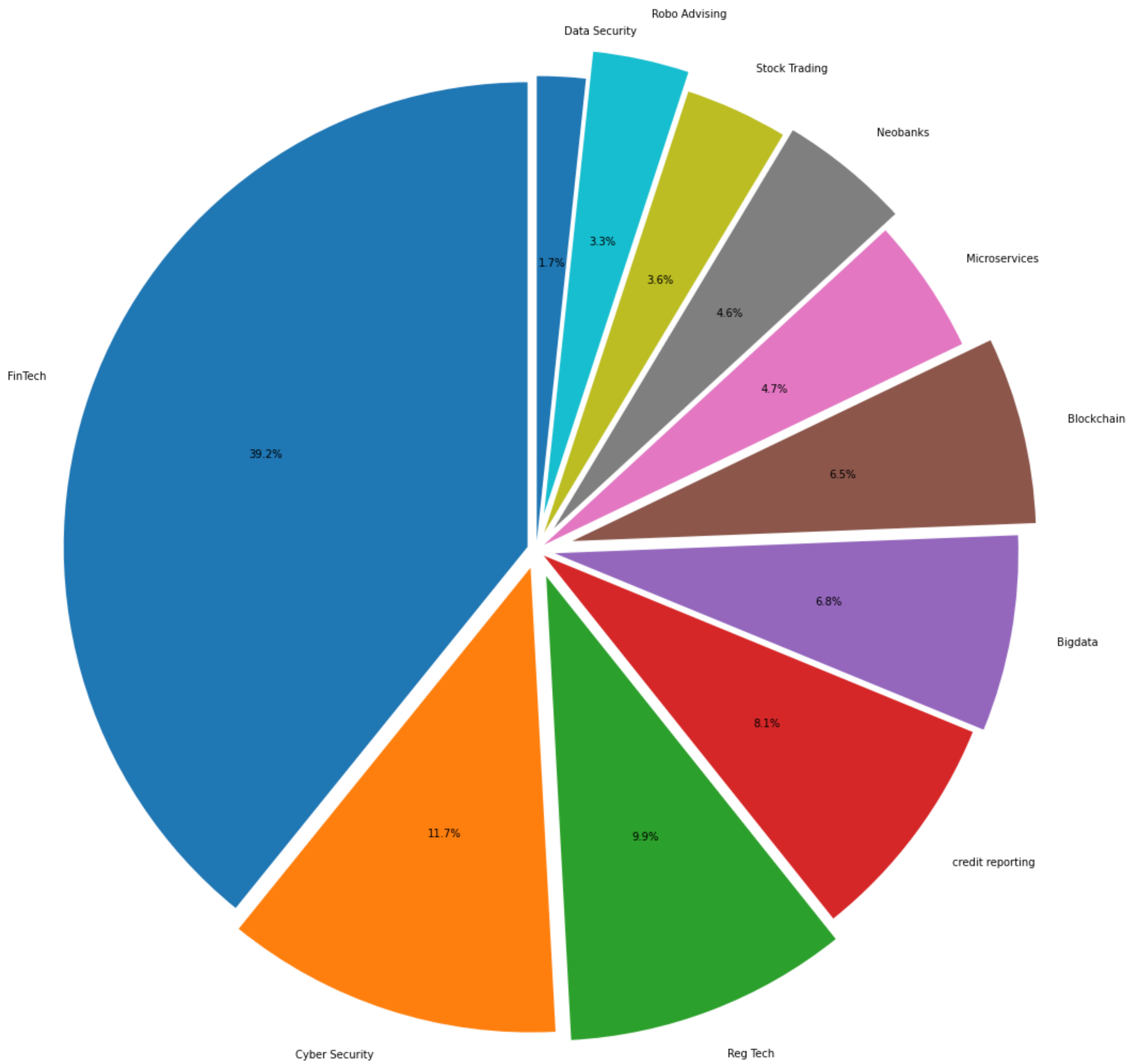
In [25]:
```
plt.figure(figsize=(20,15))
sns.countplot(train_data.Target)
plt.show()
```



From above it is quite evident that FinTech dataset is most used in the model

In [26]:
```
plt.pie(train_data.Target.value_counts(),explode=[0.1,0.2,0.3,0.1,0.2,0.4,0.1,0.3,0.2,0.4,0.1],autopct='%0.1f%%',labels=train_data.Target.value_counts().index,radius=5,startangle=90)
plt.plot()
```

Out[26]: []



Analysising top 4 most occuring technology

In [27]:
```python
train_data['Text'][train_data.Target=='FinTech'][1:10].values
```

Out[27]:
```
array(['example combined regula tory compliance often',
       'considerable capital ample customer bases',
       'possibility combining open selection best best financial roof',
       'until trading goods goods limited small community direct',
       'billion people financial system doubt enormous',
       'their platform designed digital enrolment user experience analogue proc
ess branch',
       'mobile challenge approach banking innovation competition',
       'innovation against backdrop innovation investment continued growth priv
ate sector',
       'payment method charge user making payment merchant'], dtype=object)
```

In [28]:
```python
train_data['Text'][train_data.Target=='Bigdata'][1:10].values
```

Out[28]:
```
array(['modern customer support tools provide wide variety data concerning user
requests timelines resolution customer problems',
       'emerging markets primary beneficiaries seldom established credit regist
ry',
       'according estimate third globally stored information form alphanumeric
text still image data format useful data applications',
       'since teradata added unstructured data types including json avro',
       'world technological capita capacity store information roughly doubled e
very months since every',
       'results research used product process improvements business',
       'methodology addresses handling data terms useful permutations data sour
ces complexity interrelationships difficulty deleting modifying individual reco
rds',
       'this company recently suffered series defaults',
       'suggested nick couldry joseph turow practitioners media advertising app
roach data many actionable points information millions individuals'],
      dtype=object)
```

In [29]:
```python
train_data['Text'][train_data.Target=='Cyber Security'][:10].values
```

Out[29]:
```
array(['easily access research mobile capitalize investment quickly banking mob
ile banking track investment',
       'report assessment payment prone suffer higher exposure',
       'boundary sandbox evaluation criteria after period duration sandbox diff
er case case basis',
       'this ethical financial dealing currency unable fully verify identity',
       'original bill prepared committee government',
       'must remain impossible completely secure technology',
       'after vehemently breach backed following statement',
       'financial sector include banking payment',
       'addition sole operator mobile application recently national common mobi
lity card card scheme',
       'domestic standard setting testing component resilience'],
      dtype=object)
```

In [30]:
```
train_data['Text'][train_data.Target=='Reg Tech'][:10].values
```

Out[30]:
```
array(['fact caused reevaluation regulatory approaches China',
       'Hannah Augur Regtech Buzzword Turning Heads DATACONOMY dataconomy',
       'Additionally FSOC issue recommendations primary financial regulatory ag
encies apply heightened standards financial activity practice conducted compani
es predominantly engaged financial activities',
       'Today course majority securities trading involves computers example Her
statt risk cross currency settlement risk Long Dark Shadow Herstatt ECONOMIST A
pril',
       'example introduction deposit insurance scheme China provide safety allo
wing potential failure banks',
       'Enhancing Corporate Governance RegTech promotes good corporate practice
compliance manage ment enhances desired regulatory compliance outcomes',
       'vision builds Andy Haldane whereby financial institutions regulators mo
nitor analyze real time financial information parts global financial sector und
erpin safer efficient financial system',
       'This includes test durations milestones risk analysis investigation pot
ential exposure measurement metrics exit strategy',
       'argue Part regulatory requirements also necessitate ever increasing app
lication technology regulators order monitor rivers data sent',
       'example highlights RegTech Report real time settlement could achieved a
utomation global consensus blockchain'],
      dtype=object)
```

In [31]:
```
class Lemmatizer():
    def __init__(self):
        self.lemmatizer=WordNetLemmatizer()
    def __call__(self,sentence):
        sentence=re.sub('(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]
*)',' ',sentence)
        sentence=re.sub('[^0-9a-z]',' ',sentence)

        return [self.lemmatizer.lemmatize(word) for word in sentence.split() if
word not in stopwords.words('english') if len(self.lemmatizer.lemmatize(word))>
1]
```

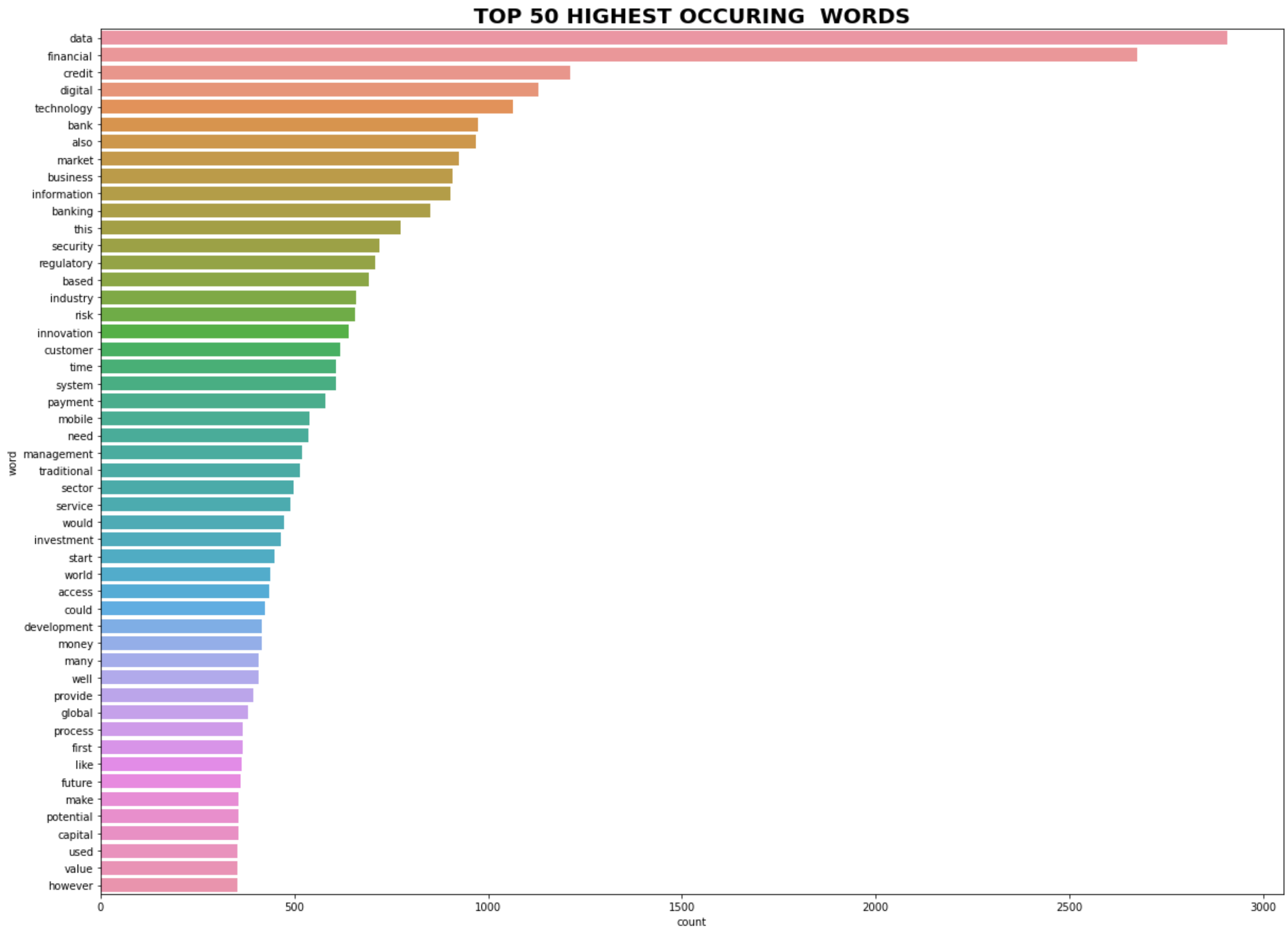
In [32]:
```
token=Tokenizer(num_words=3000,oov_token=Lemmatizer())
```

In [33]:
```
token.fit_on_texts(train_data.Text)
```

In [ ]:
```
# token.word_index
```

In [34]:
```
p=sorted(token.word_counts.items(), key=lambda item : item[1],reverse=True)
df=pd.DataFrame(columns=['word','count'])
i=1
for k,v in p:
    df2=pd.DataFrame({'word':[k],'count':[v]})
    df=df.append(df2)
    if i==50:
        break
    i+=1
```

In [35]:
```
plt.figure(figsize=(20,15))
sns.barplot(df['count'],df['word'])
plt.title('TOP 50 HIGHEST OCCURING  WORDS',fontdict={'size':20,'weight':'bold'
})
plt.show()
```



In [36]:
```python
wc=WordCloud()
wc.generate(' '.join(word for i,word in enumerate(token.word_index.keys()) if i
>1))
plt.figure(figsize=(20,10))
plt.axis('off')
plt.title('Word Cloud of top 200 words',fontdict={'size':30,'weight':'bold'})
plt.imshow(wc)
```

Out[36]:
```
<matplotlib.image.AxesImage at 0x7ff4726ef0d0>
```

Word Cloud of top 200 words

In [37]:
```python
train_text=token.texts_to_sequences(train_data.Text)
```

In [38]:
```python
train_text_padded=pad_sequences(train_text,maxlen=30,padding='post',truncating=
'pre')
```

In [39]:
```python
lbl_encoder=LabelEncoder()
train_target=lbl_encoder.fit_transform(train_data.Target)
```

In [40]:
```python
val_text=token.texts_to_sequences(test_data.Text)
val_text_padded=pad_sequences(val_text,maxlen=30,padding='post',truncating='pr
e')
val_target=lbl_encoder.transform(test_data.Target)
```

In [41]:
```python
V=len(token.word_index)
```

Creating Callback function

In [42]:
```
early_stop=EarlyStopping(patience=6)
reduceLR=ReduceLROnPlateau(patience=4)
```

Creating simple LSTM model with 3 Dense layers

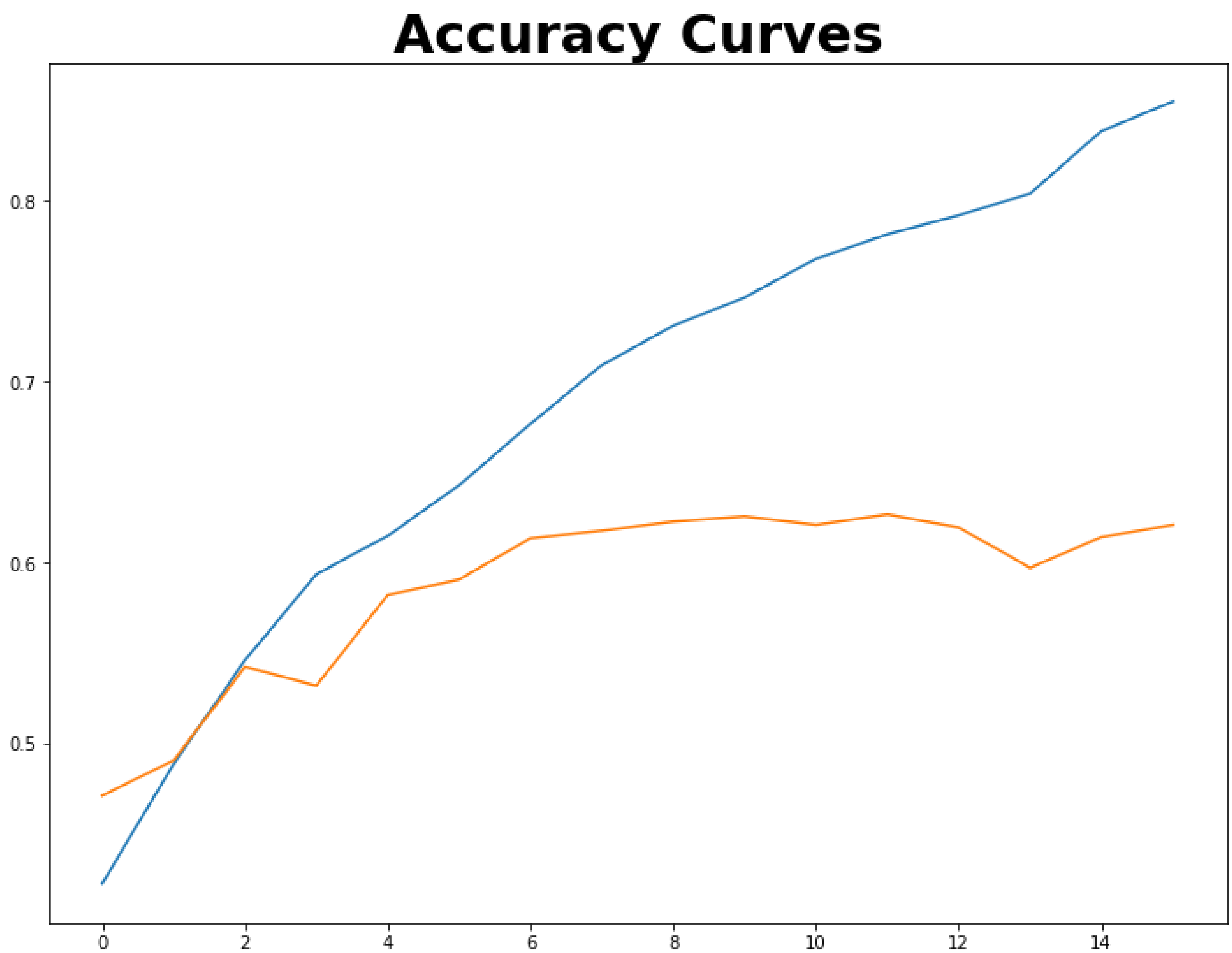
In [43]:
```
embedding_feature=32
model1=Sequential()
model1.add(Embedding(V+1,embedding_feature,input_shape=(30,)))
model1.add(LSTM(64))
model1.add(Dense(512,activation='relu'))
model1.add(Dense(128,activation='relu'))
model1.add(Dense(11,activation='softmax'))
# model.add()
```

In [45]:
```
model1.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=
['accuracy'])
```

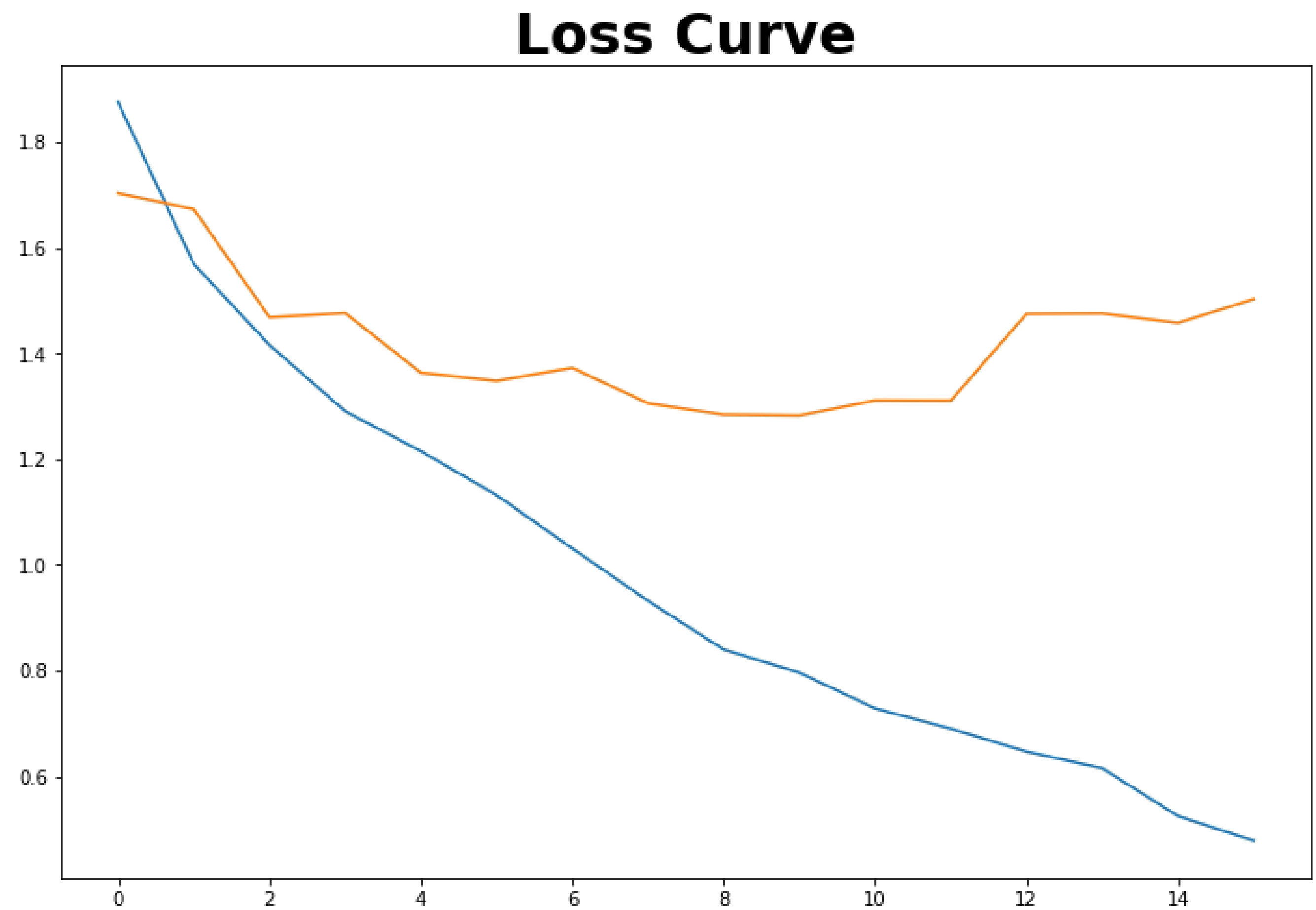
In [47]:
```
r=model1.fit(train_text_padded,train_target,validation_data=(val_text_padded,va
l_target),epochs=100,batch_size=50,callbacks=[early_stop,reduceLR])
```

```
Epoch 1/100
319/319 [==============================] - 5s 8ms/step - loss: 2.0269 - accurac
y: 0.3936 - val_loss: 1.7033 - val_accuracy: 0.4710
Epoch 2/100
319/319 [==============================] - 3s 8ms/step - loss: 1.5728 - accurac
y: 0.4895 - val_loss: 1.6734 - val_accuracy: 0.4906
Epoch 3/100
319/319 [==============================] - 2s 7ms/step - loss: 1.4225 - accurac
y: 0.5427 - val_loss: 1.4687 - val_accuracy: 0.5422
Epoch 4/100
319/319 [==============================] - 2s 7ms/step - loss: 1.3243 - accurac
y: 0.5856 - val_loss: 1.4768 - val_accuracy: 0.5319
Epoch 5/100
319/319 [==============================] - 2s 7ms/step - loss: 1.2388 - accurac
y: 0.6061 - val_loss: 1.3637 - val_accuracy: 0.5821
Epoch 6/100
319/319 [==============================] - 2s 7ms/step - loss: 1.1354 - accurac
y: 0.6448 - val_loss: 1.3484 - val_accuracy: 0.5907
Epoch 7/100
319/319 [==============================] - 3s 8ms/step - loss: 1.0253 - accurac
y: 0.6787 - val_loss: 1.3729 - val_accuracy: 0.6135
Epoch 8/100
319/319 [==============================] - 2s 7ms/step - loss: 0.9171 - accurac
y: 0.7153 - val_loss: 1.3059 - val_accuracy: 0.6177
Epoch 9/100
319/319 [==============================] - 2s 7ms/step - loss: 0.8341 - accurac
y: 0.7324 - val_loss: 1.2844 - val_accuracy: 0.6227
Epoch 10/100
319/319 [==============================] - 3s 9ms/step - loss: 0.7764 - accurac
y: 0.7534 - val_loss: 1.2828 - val_accuracy: 0.6256
Epoch 11/100
319/319 [==============================] - 2s 8ms/step - loss: 0.6999 - accurac
y: 0.7772 - val_loss: 1.3112 - val_accuracy: 0.6209
Epoch 12/100
319/319 [==============================] - 2s 7ms/step - loss: 0.6608 - accurac
y: 0.7924 - val_loss: 1.3109 - val_accuracy: 0.6266
Epoch 13/100
319/319 [==============================] - 2s 7ms/step - loss: 0.6304 - accurac
y: 0.7971 - val_loss: 1.4753 - val_accuracy: 0.6195
Epoch 14/100
319/319 [==============================] - 2s 7ms/step - loss: 0.5894 - accurac
y: 0.8176 - val_loss: 1.4761 - val_accuracy: 0.5971
Epoch 15/100
319/319 [==============================] - 2s 7ms/step - loss: 0.5558 - accurac
y: 0.8298 - val_loss: 1.4581 - val_accuracy: 0.6142
Epoch 16/100
319/319 [==============================] - 3s 8ms/step - loss: 0.4957 - accurac
y: 0.8516 - val_loss: 1.5030 - val_accuracy: 0.6209
```

In [51]:
```
plt.figure(figsize=(12,9))
plt.plot(r.history['accuracy'])
plt.plot(r.history['val_accuracy'])
plt.title('Accuracy Curves',fontdict={'size':30,'weight':'bold'})
plt.show()
```



In [52]:
```
plt.figure(figsize=(12,8))
plt.plot(r.history['loss'])
plt.plot(r.history['val_loss'])
plt.title('Loss Curve',fontdict={'size':30,'weight':'bold'})
plt.show()
```



In [54]:
```
print('model 1 train',classification_report(train_target,model1.predict_classes
(train_text_padded)))
print('model 1 test',classification_report(val_target,model1.predict_classes(va
l_text_padded)))
```

```
model 1 train               precision    recall  f1-score   support

           0       0.94      0.96      0.95      1084
           1       0.91      0.82      0.87      1033
           2       0.87      0.81      0.84      1859
           3       0.61      0.58      0.59       269
           4       0.83      0.92      0.87      6228
           5       0.96      0.81      0.88       750
           6       0.73      0.60      0.66       724
           7       0.98      0.97      0.98      1570
           8       0.80      0.72      0.76       532
           9       0.94      0.77      0.85       569
          10       0.85      0.83      0.84      1283

    accuracy                           0.86     15901
   macro avg       0.86      0.80      0.82     15901
weighted avg       0.86      0.86      0.86     15901

model 1 test               precision    recall  f1-score   support

           0       0.66      0.68      0.67       191
           1       0.54      0.47      0.50       182
           2       0.56      0.54      0.55       328
           3       0.17      0.15      0.16        48
           4       0.65      0.74      0.69      1099
           5       0.57      0.44      0.50       133
           6       0.28      0.18      0.22       128
           7       0.83      0.84      0.84       277
           8       0.42      0.29      0.34        94
           9       0.67      0.44      0.53       101
          10       0.59      0.64      0.62       226

    accuracy                           0.62      2807
   macro avg       0.54      0.49      0.51      2807
weighted avg       0.61      0.62      0.61      2807
```

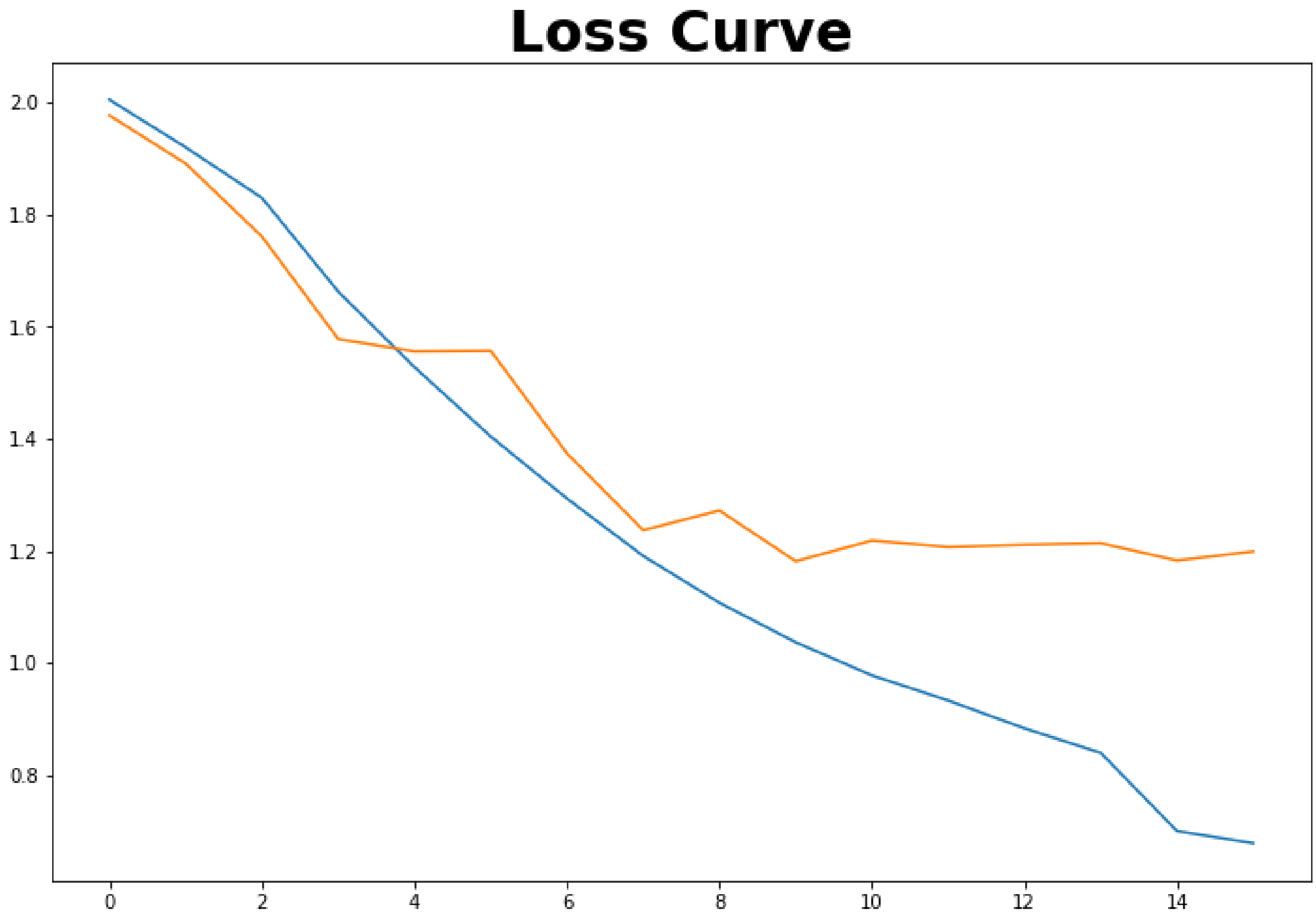
Creating a LSTM model with 3 dense layer and GlobalMaxPooling1D with SGD optimizer

In [55]:
```
embedding_feature=32
model2=Sequential()
model2.add(Embedding(V+1,embedding_feature,input_shape=(30,)))
model2.add(LSTM(64,return_sequences=True))
model2.add(GlobalMaxPooling1D())
model2.add(Dense(512))
model2.add(Dense(128))
model2.add(Dense(11,activation='softmax'))
# model.add()
```

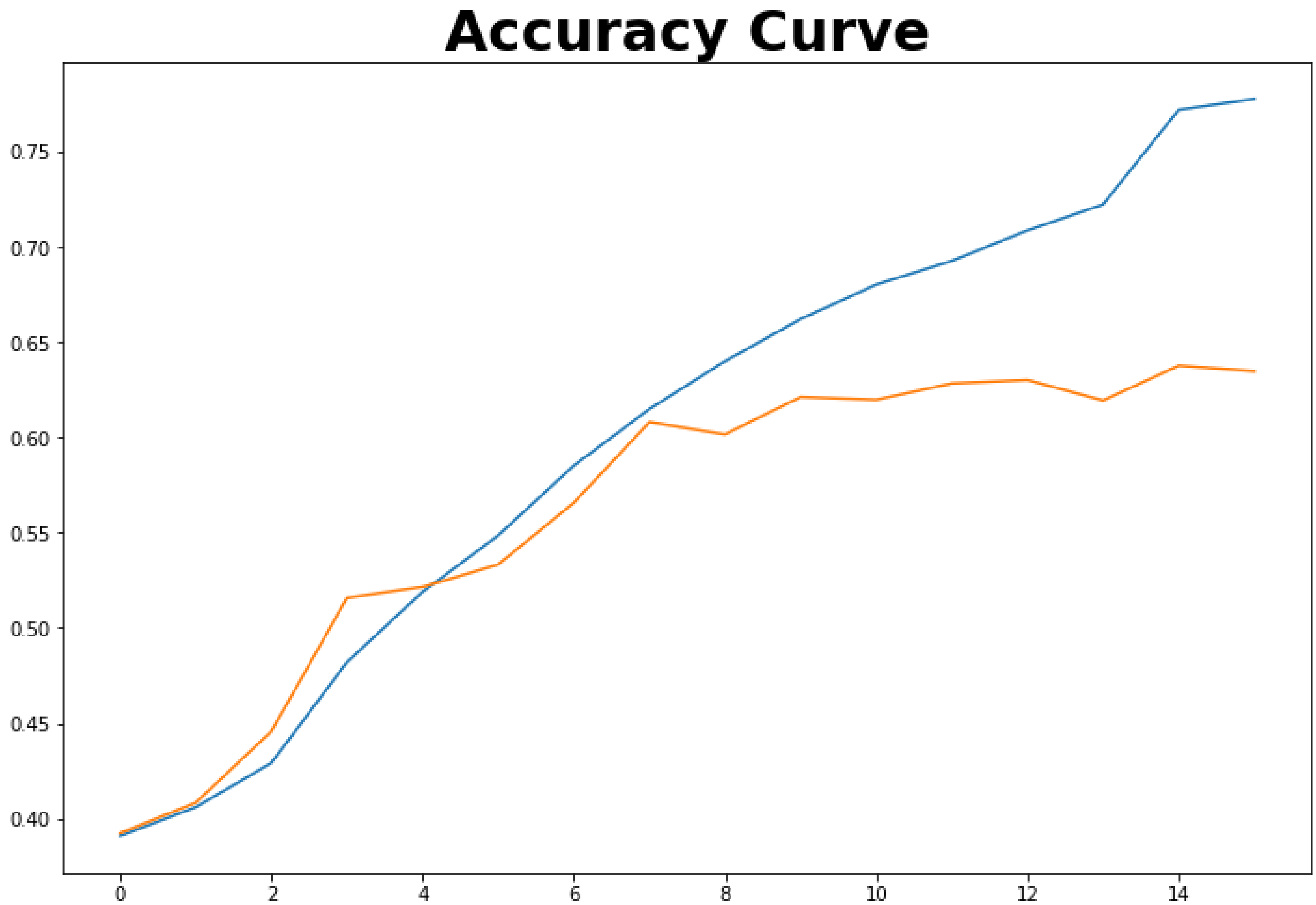
In [56]:
```
model2.compile(optimizer=keras.optimizers.SGD(0.1,0.009),loss='sparse_categoric
al_crossentropy',metrics=['accuracy'])
r2=model2.fit(train_text_padded,train_target,validation_data=(val_text_padded,v
al_target),epochs=100,callbacks=[early_stop,reduceLR])
```

```
Epoch 1/100
497/497 [==============================] - 4s 5ms/step - loss: 2.0074 - accurac
y: 0.3961 - val_loss: 1.9767 - val_accuracy: 0.3922
Epoch 2/100
497/497 [==============================] - 2s 5ms/step - loss: 1.9529 - accurac
y: 0.3961 - val_loss: 1.8908 - val_accuracy: 0.4083
Epoch 3/100
497/497 [==============================] - 2s 5ms/step - loss: 1.8507 - accurac
y: 0.4225 - val_loss: 1.7606 - val_accuracy: 0.4457
Epoch 4/100
497/497 [==============================] - 3s 5ms/step - loss: 1.7222 - accurac
y: 0.4585 - val_loss: 1.5778 - val_accuracy: 0.5159
Epoch 5/100
497/497 [==============================] - 2s 5ms/step - loss: 1.5366 - accurac
y: 0.5199 - val_loss: 1.5559 - val_accuracy: 0.5216
Epoch 6/100
497/497 [==============================] - 2s 5ms/step - loss: 1.4280 - accurac
y: 0.5437 - val_loss: 1.5569 - val_accuracy: 0.5333
Epoch 7/100
497/497 [==============================] - 2s 5ms/step - loss: 1.3133 - accurac
y: 0.5757 - val_loss: 1.3737 - val_accuracy: 0.5657
Epoch 8/100
497/497 [==============================] - 2s 5ms/step - loss: 1.2064 - accurac
y: 0.6094 - val_loss: 1.2368 - val_accuracy: 0.6081
Epoch 9/100
497/497 [==============================] - 3s 5ms/step - loss: 1.0987 - accurac
y: 0.6425 - val_loss: 1.2719 - val_accuracy: 0.6017
Epoch 10/100
497/497 [==============================] - 2s 5ms/step - loss: 1.0243 - accurac
y: 0.6687 - val_loss: 1.1816 - val_accuracy: 0.6213
Epoch 11/100
497/497 [==============================] - 2s 5ms/step - loss: 0.9508 - accurac
y: 0.6911 - val_loss: 1.2183 - val_accuracy: 0.6199
Epoch 12/100
497/497 [==============================] - 2s 5ms/step - loss: 0.9122 - accurac
y: 0.6983 - val_loss: 1.2069 - val_accuracy: 0.6284
Epoch 13/100
497/497 [==============================] - 3s 6ms/step - loss: 0.8731 - accurac
y: 0.7126 - val_loss: 1.2109 - val_accuracy: 0.6302
Epoch 14/100
497/497 [==============================] - 2s 5ms/step - loss: 0.8041 - accurac
y: 0.7401 - val_loss: 1.2136 - val_accuracy: 0.6195
Epoch 15/100
497/497 [==============================] - 2s 5ms/step - loss: 0.7087 - accurac
y: 0.7702 - val_loss: 1.1829 - val_accuracy: 0.6377
Epoch 16/100
497/497 [==============================] - 2s 5ms/step - loss: 0.6663 - accurac
y: 0.7823 - val_loss: 1.1988 - val_accuracy: 0.6348
```

In [57]:
```
plt.figure(figsize=(12,8))
plt.plot(r2.history['loss'])
plt.plot(r2.history['val_loss'])
plt.title('Loss Curve',fontdict={'size':30,'weight':'bold'})
plt.show()
```



In [58]:
```
plt.figure(figsize=(12,8))
plt.plot(r2.history['accuracy'])
plt.plot(r2.history['val_accuracy'])
plt.title('Accuracy Curve',fontdict={'size':30,'weight':'bold'})
plt.show()
```



```
In [59]: print('model 2 train',classification_report(train_target,model2.predict_classes(train_text_padded)))
print('model 2 test',classification_report(val_target,model2.predict_classes(val_text_padded)))
```

```
model 2 train               precision    recall  f1-score   support

           0       0.88      0.88      0.88      1084
           1       0.82      0.68      0.74      1033
           2       0.74      0.71      0.72      1859
           3       0.55      0.14      0.22       269
           4       0.76      0.91      0.82      6228
           5       0.80      0.78      0.79       750
           6       0.70      0.21      0.33       724
           7       0.93      0.93      0.93      1570
           8       0.72      0.54      0.62       532
           9       0.78      0.77      0.78       569
          10       0.80      0.73      0.76      1283

    accuracy                           0.79     15901
   macro avg       0.77      0.66      0.69     15901
weighted avg       0.78      0.79      0.77     15901

model 2 test               precision    recall  f1-score   support

           0       0.69      0.69      0.69       191
           1       0.49      0.37      0.43       182
           2       0.54      0.55      0.54       328
           3       0.18      0.06      0.09        48
           4       0.64      0.78      0.70      1099
           5       0.66      0.52      0.58       133
           6       0.33      0.11      0.16       128
           7       0.82      0.84      0.83       277
           8       0.50      0.31      0.38        94
           9       0.69      0.56      0.62       101
          10       0.67      0.62      0.64       226

    accuracy                           0.63      2807
   macro avg       0.56      0.49      0.52      2807
weighted avg       0.62      0.63      0.62      2807
```

Creating a LSTM model with 3 dense layer and GlobalMaxPooling1D with SGD optimizer and relu activation on each layer of dense layer.

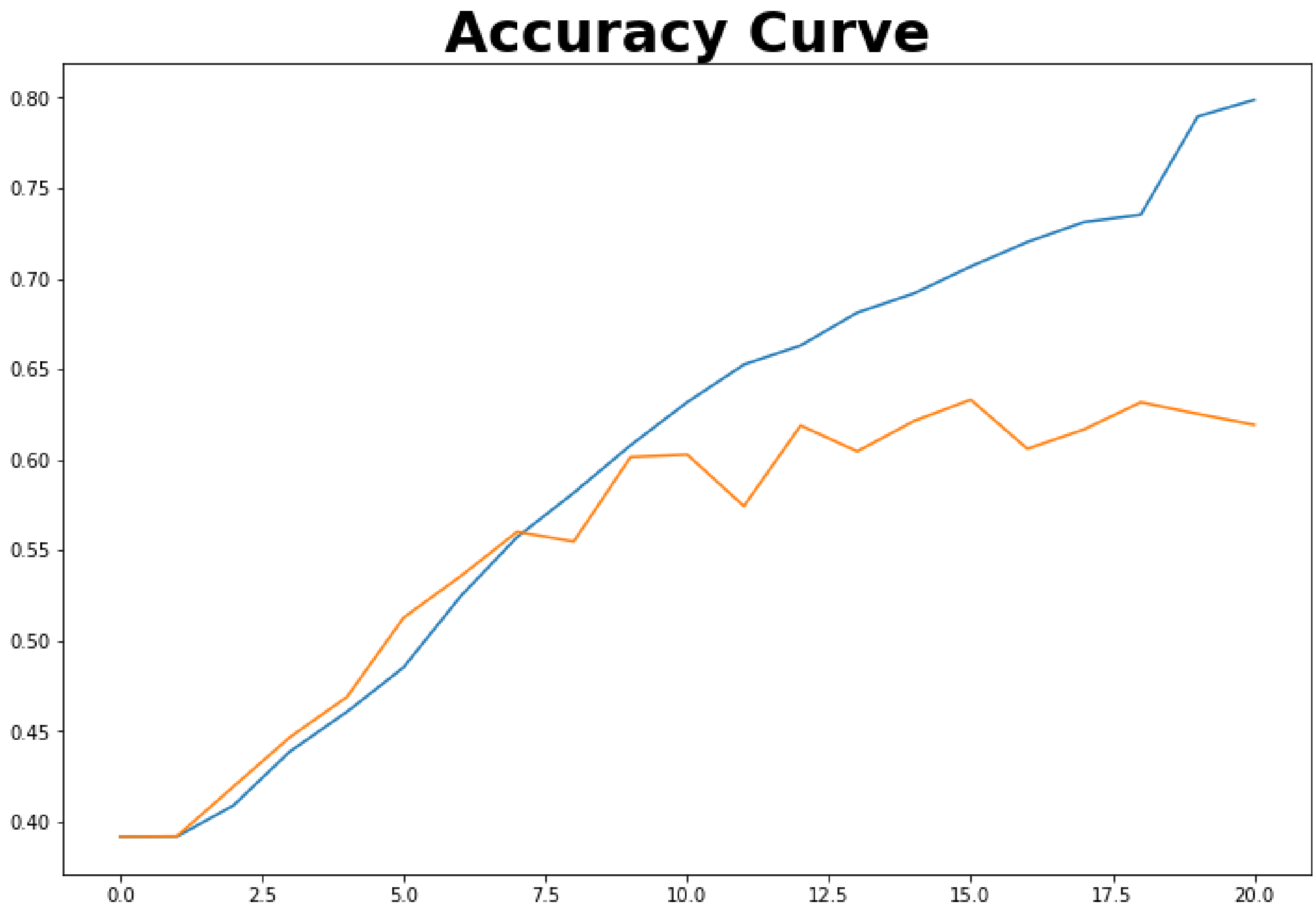
```
In [60]: embedding_feature=32
model3=Sequential()
model3.add(Embedding(V+1,embedding_feature,input_shape=(30,)))
model3.add(LSTM(64,return_sequences=True))
model3.add(GlobalMaxPooling1D())
model3.add(Dense(512,activation='relu'))
model3.add(Dense(128,activation='relu'))
model3.add(Dense(11,activation='softmax'))
# model.add()
```

In [61]:
```
model3.compile(optimizer=keras.optimizers.SGD(0.1,momentum=0.09),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
r3=model3.fit(train_text_padded,train_target,validation_data=(val_text_padded,val_target),epochs=100,callbacks=[early_stop,reduceLR])
```

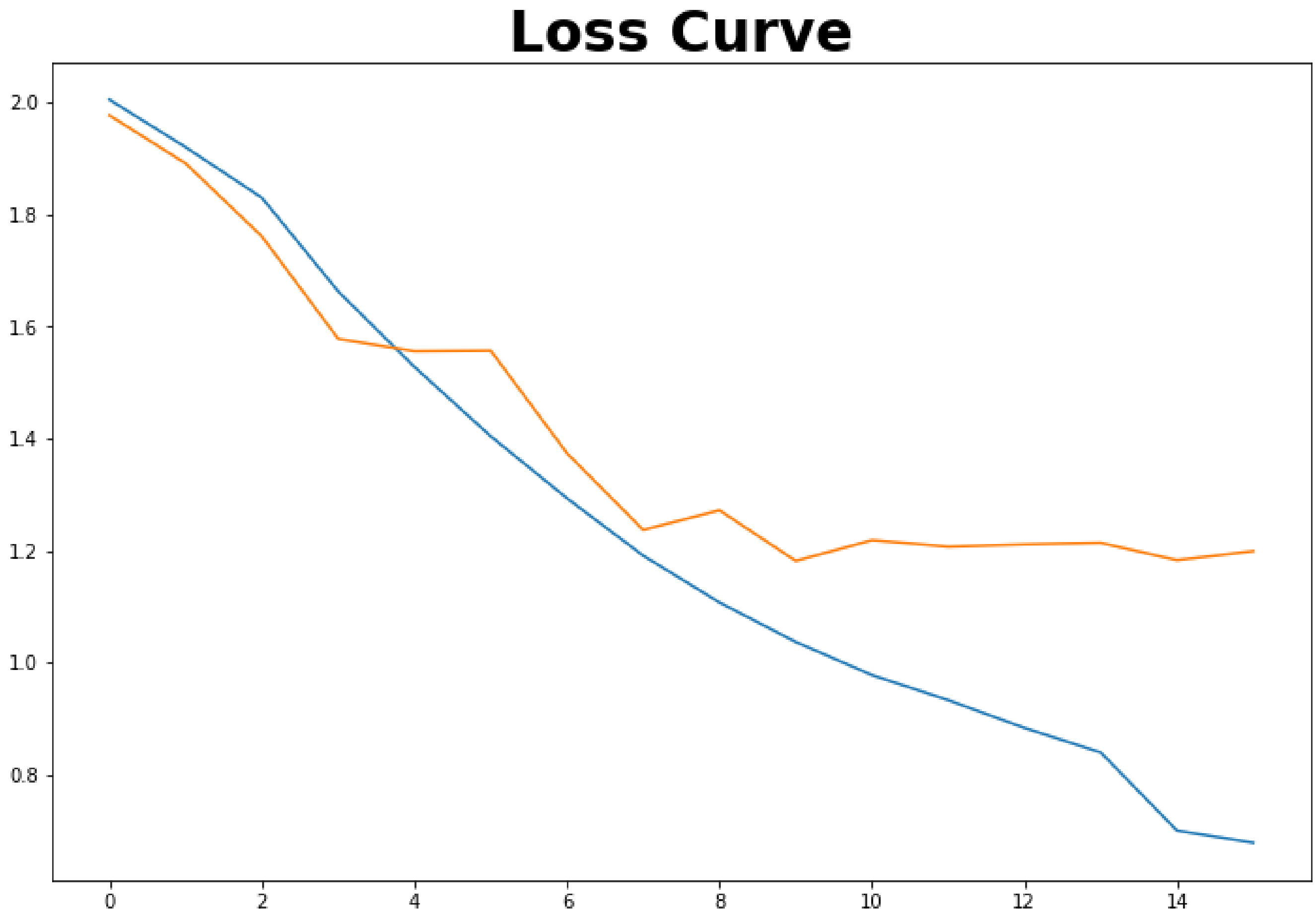
```
Epoch 1/100
497/497 [==============================] - 4s 6ms/step - loss: 2.0291 - accurac
y: 0.3914 - val_loss: 2.0013 - val_accuracy: 0.3915
Epoch 2/100
497/497 [==============================] - 2s 5ms/step - loss: 1.9923 - accurac
y: 0.3922 - val_loss: 1.9630 - val_accuracy: 0.3919
Epoch 3/100
497/497 [==============================] - 3s 5ms/step - loss: 1.9269 - accurac
y: 0.4051 - val_loss: 1.8659 - val_accuracy: 0.4193
Epoch 4/100
497/497 [==============================] - 2s 5ms/step - loss: 1.8238 - accurac
y: 0.4335 - val_loss: 1.8074 - val_accuracy: 0.4467
Epoch 5/100
497/497 [==============================] - 2s 5ms/step - loss: 1.7418 - accurac
y: 0.4554 - val_loss: 1.6873 - val_accuracy: 0.4688
Epoch 6/100
497/497 [==============================] - 2s 5ms/step - loss: 1.6433 - accurac
y: 0.4809 - val_loss: 1.5565 - val_accuracy: 0.5126
Epoch 7/100
497/497 [==============================] - 3s 5ms/step - loss: 1.5175 - accurac
y: 0.5230 - val_loss: 1.4625 - val_accuracy: 0.5354
Epoch 8/100
497/497 [==============================] - 2s 5ms/step - loss: 1.4099 - accurac
y: 0.5545 - val_loss: 1.3864 - val_accuracy: 0.5600
Epoch 9/100
497/497 [==============================] - 2s 5ms/step - loss: 1.3205 - accurac
y: 0.5741 - val_loss: 1.3542 - val_accuracy: 0.5547
Epoch 10/100
497/497 [==============================] - 2s 5ms/step - loss: 1.2312 - accurac
y: 0.6018 - val_loss: 1.2743 - val_accuracy: 0.6014
Epoch 11/100
497/497 [==============================] - 2s 5ms/step - loss: 1.1481 - accurac
y: 0.6296 - val_loss: 1.3174 - val_accuracy: 0.6028
Epoch 12/100
497/497 [==============================] - 3s 6ms/step - loss: 1.0789 - accurac
y: 0.6562 - val_loss: 1.3329 - val_accuracy: 0.5743
Epoch 13/100
497/497 [==============================] - 2s 5ms/step - loss: 1.0208 - accurac
y: 0.6672 - val_loss: 1.2069 - val_accuracy: 0.6188
Epoch 14/100
497/497 [==============================] - 2s 5ms/step - loss: 0.9612 - accurac
y: 0.6833 - val_loss: 1.2489 - val_accuracy: 0.6046
Epoch 15/100
497/497 [==============================] - 2s 5ms/step - loss: 0.9139 - accurac
y: 0.6936 - val_loss: 1.2059 - val_accuracy: 0.6213
Epoch 16/100
497/497 [==============================] - 3s 5ms/step - loss: 0.8652 - accurac
y: 0.7187 - val_loss: 1.2074 - val_accuracy: 0.6331
Epoch 17/100
497/497 [==============================] - 2s 5ms/step - loss: 0.8345 - accurac
y: 0.7260 - val_loss: 1.2399 - val_accuracy: 0.6060
Epoch 18/100
497/497 [==============================] - 2s 5ms/step - loss: 0.7967 - accurac
y: 0.7383 - val_loss: 1.2284 - val_accuracy: 0.6167
Epoch 19/100
497/497 [==============================] - 2s 5ms/step - loss: 0.7635 - accurac
y: 0.7417 - val_loss: 1.2479 - val_accuracy: 0.6316
Epoch 20/100
497/497 [==============================] - 2s 5ms/step - loss: 0.6505 - accurac
y: 0.7879 - val_loss: 1.2571 - val_accuracy: 0.6252
Epoch 21/100
```

```
497/497 [==============================] - 3s 5ms/step - loss: 0.6223 - accuracy: 0.8018 - val_loss: 1.2708 - val_accuracy: 0.6192
```

In [62]:
```python
plt.figure(figsize=(12,8))
plt.plot(r3.history['accuracy'])
plt.plot(r3.history['val_accuracy'])
plt.title('Accuracy Curve',fontdict={'size':30,'weight':'bold'})
plt.show()
```



In [63]:
```
plt.figure(figsize=(12,8))
plt.plot(r2.history['loss'])
plt.plot(r2.history['val_loss'])
plt.title('Loss Curve',fontdict={'size':30,'weight':'bold'})
plt.show()
```



In [64]:
```
print('model 3 train',classification_report(train_target,model3.predict_classes
(train_text_padded)))
print('model 3 test',classification_report(val_target,model3.predict_classes(va
l_text_padded)))
```

```
model 3 train               precision    recall  f1-score   support

           0       0.86      0.94      0.90      1084
           1       0.86      0.74      0.79      1033
           2       0.77      0.70      0.73      1859
           3       0.50      0.21      0.30       269
           4       0.78      0.90      0.84      6228
           5       0.86      0.73      0.79       750
           6       0.58      0.39      0.47       724
           7       0.97      0.91      0.94      1570
           8       0.76      0.60      0.67       532
           9       0.82      0.82      0.82       569
          10       0.81      0.77      0.79      1283

    accuracy                           0.80     15901
   macro avg       0.78      0.70      0.73     15901
weighted avg       0.80      0.80      0.80     15901

model 3 test              precision    recall  f1-score   support

           0       0.65      0.72      0.69       191
           1       0.48      0.42      0.45       182
           2       0.57      0.53      0.55       328
           3       0.29      0.15      0.19        48
           4       0.64      0.75      0.69      1099
           5       0.56      0.46      0.50       133
           6       0.28      0.20      0.23       128
           7       0.86      0.81      0.83       277
           8       0.39      0.26      0.31        94
           9       0.59      0.50      0.54       101
          10       0.63      0.62      0.62       226

    accuracy                           0.62      2807
   macro avg       0.54      0.49      0.51      2807
weighted avg       0.61      0.62      0.61      2807
```

Trying Simple Neural Network or Logistic Regresion Prediction

In [65]:
```
counter=CountVectorizer(max_features=5000,tokenizer=Lemmatizer())
train_text_seq=counter.fit_transform(train_data.Text).toarray()
```

In [66]:
```
feature_names=counter.get_feature_names()
```

In [67]:
```
wc_counter=WordCloud()
wc_counter.generate(' '.join(word for word in feature_names))
plt.figure(figsize=(20,10))
plt.axis('off')
plt.title('Word Cloud of 200 words of counter vectorizer',fontdict={'size':30,
'weight':'bold'})
plt.imshow(wc_counter)
```

Out[67]:
```
<matplotlib.image.AxesImage at 0x7ff3834964d0>
```

Word Cloud of 200 words of counter vectorizer

In [68]:
```
test_text_seq=counter.transform(test_data.Text).toarray()
```

In [69]:
```
model4=LogisticRegression(max_iter=5000)
model4.fit(train_text_seq,train_target)
```

Out[69]:
```
LogisticRegression(max_iter=5000)
```

In [70]:
```
print('model 4 train',classification_report(train_target,model4.predict(train_text_seq)))
print('model 4 test',classification_report(val_target,model4.predict(test_text_seq)))
```

```
model 4 train               precision    recall  f1-score   support

           0       0.95      0.89      0.92      1084
           1       0.93      0.83      0.88      1033
           2       0.89      0.78      0.83      1859
           3       0.94      0.62      0.75       269
           4       0.81      0.96      0.88      6228
           5       0.97      0.84      0.90       750
           6       0.93      0.68      0.79       724
           7       0.98      0.94      0.96      1570
           8       0.92      0.78      0.85       532
           9       0.96      0.83      0.89       569
          10       0.93      0.83      0.88      1283

    accuracy                           0.88     15901
   macro avg       0.93      0.82      0.87     15901
weighted avg       0.89      0.88      0.88     15901

model 4 test               precision    recall  f1-score   support

           0       0.68      0.60      0.64       191
           1       0.66      0.49      0.56       182
           2       0.58      0.54      0.56       328
           3       0.36      0.19      0.25        48
           4       0.63      0.84      0.72      1099
           5       0.73      0.51      0.60       133
           6       0.55      0.28      0.37       128
           7       0.89      0.78      0.83       277
           8       0.62      0.35      0.45        94
           9       0.79      0.53      0.64       101
          10       0.73      0.64      0.68       226

    accuracy                           0.66      2807
   macro avg       0.66      0.52      0.57      2807
weighted avg       0.67      0.66      0.65      2807
```

In [71]:
```
lbl_encoder.classes_
```

Out[71]:
```
array(['Bigdata', 'Blockchain', 'Cyber Security', 'Data Security',
       'FinTech', 'Microservices', 'Neobanks', 'Reg Tech',
       'Robo Advising', 'Stock Trading', 'credit reporting'], dtype=object)
```

Model Selection

Best accuracy we obtained is for logistic model. Through model obtained a good accuracy but accuracy in prediction of data security and Robo-Advising is less , as there are not sufficient data of this two instances. This could be considered as disadvantage of this model

Dumping the best model for further use

In [72]:
```
import pickle
```

In [74]:
```
with open('log_model.pickle','wb') as handle:
    pickle.dump(model4,handle,protocol=pickle.HIGHEST_PROTOCOL)

with open('log_tokenizer.pickle','wb') as handle:
    pickle.dump(counter,handle,protocol=pickle.HIGHEST_PROTOCOL)

with open('log_encoder.pickle','wb') as handle:
    pickle.dump(lbl_encoder,handle,protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]: