LSTM classifier for abusive/sarcastic language

Import libraries

In [3]:
```
import pandas as pd
import numpy as np

from pymongo import MongoClient
from Preprocessing import config
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from keras.utils.np_utils import to_categorical
```

```
c:\users\evaarevalo\appdata\local\programs\python\python36\lib\site-packages
\h5py\__init__.py:36: FutureWarning: Conversion of the second argument of iss
ubdtype from `float` to `np.floating` is deprecated. In future, it will be tr
eated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

MongoDB connection

In [4]:
```
client = MongoClient(config.MONGODB['hostname'], config.MONGODB['port'])
db = client[config.MONGODB['db']]
collection = db[config.MONGODB['collection_news_and_sarcasm']]
```

Get datasets

In [5]:
```
def getDatasetsFromMongoDB():
        ''' mongodb to pandas dataframe, export to csv and return'''
        results=collection.find()
        #strip and reshuflle
        df =  pd.DataFrame(list(results))
        df=df[['label','text']]
        df=df.reindex(np.random.permutation(df.index))
        df.to_csv('sarcasm_and_news_dataset.csv',encoding='utf-8-sig')
        return df
```

In [13]:
```
def getDatasetsFromCsv():
        '''import csv, reshuffle and return it'''
        df=pd.read_csv('sarcasm_and_news_dataset.csv')
        df=df.reindex(np.random.permutation(df.index))
        return df[['label','text']]
```

In [14]:
```
data=getDatasetsFromCsv()
train,test = train_test_split(data,test_size=0.1)
```

Embedding layer parameters

In [20]:
```
max_features=2000
embedding_dim=128
lstm_out_dim=196
batch_size=32
```

Tokenize

In [16]:
```
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)
```

Build model

In [17]:
```
model = Sequential()
model.add(Embedding(max_features, embedding_dim,input_length=X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out_dim, dropout=0.2,recurrent_dropout=0.2))
model.add(Dense(2,activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 41, 128)           256000
_________________________________________________________________
spatial_dropout1d_2 (Spatial (None, 41, 128)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 196)               254800
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 394
=================================================================
Total params: 511,194
Trainable params: 511,194
Non-trainable params: 0
_________________________________________________________________
None
```

Train

In [21]:
```
Y= pd.get_dummies(data['label']).values
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.33,
    random_state=42)
#Get shapes
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
model.fit(X_train, Y_train, epochs = 7, batch_size=batch_size, verbose=2)
```

```
(17592, 41) (17592, 2)
(8665, 41) (8665, 2)
Epoch 1/7
 - 59s - loss: 0.1216 - acc: 0.9499
Epoch 2/7
 - 57s - loss: 0.0502 - acc: 0.9822
Epoch 3/7
 - 52s - loss: 0.0357 - acc: 0.9867
Epoch 4/7
 - 54s - loss: 0.0261 - acc: 0.9909
Epoch 5/7
 - 58s - loss: 0.0204 - acc: 0.9933
Epoch 6/7
 - 58s - loss: 0.0195 - acc: 0.9936
Epoch 7/7
 - 57s - loss: 0.0135 - acc: 0.9955
```

Out[21]:
```
<keras.callbacks.History at 0x1c1630f9588>
```

Validate

In [22]:
```
validation_size = 1500

X_validate = X_test[-validation_size:]
Y_validate = Y_test[-validation_size:]
X_test = X_test[:-validation_size]
Y_test = Y_test[:-validation_size]
score,acc = model.evaluate(X_test, Y_test, verbose = 2, batch_size = batch_siz
e)

print("score: %.2f" % (score))
print("acc: %.2f" % (acc))
```

```
score: 0.07
acc: 0.98
```

In [25]:
```
positive_count=0
negative_count=0
positive_correct=0
negative_correct = 0

for x in range(len(X_validate)):

    result = model.predict(X_validate[x].reshape(1,X_test.shape[1]),batch_size
=1,verbose = 2)[0]

    if np.argmax(result) == np.argmax(Y_validate[x]):
        if np.argmax(Y_validate[x]) == 0:
            negative_correct += 1
        else:
            positive_correct += 1

    if np.argmax(Y_validate[x]) == 0:
        negative_count += 1
    else:
        positive_count += 1

print("Positive Accuracy", positive_correct/positive_count*100, "%")
print("Negative Accuracy", negative_correct/negative_count*100, "%")
```

```
Positive Accuracy 98.15100154083206 %
Negative Accuracy 97.76733254994124 %
```