# Part-2_Deep Learning Models_ NLP

September 3, 2019

References:
https://mlwhiz.com/blog/2019/01/17/deeplearning_nlp_preprocess/

In [1]:
```
import numpy as np
import pandas as pd
import plotly.offline as py
from plotly.offline import init_notebook_mode,iplot
init_notebook_mode(connected=True)
import plotly.graph_objs as go
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
import matplotlib.pyplot as plt
%matplotlib inline
from wordcloud import WordCloud
import string
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import confusion_matrix,f1_score,roc_curve,make_scorer
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB,MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from sklearn.manifold import TSNE
import os
import scikitplot as skplt
import seaborn as sns
import time
print(os.listdir("../input"))
stopwords=set(stopwords.words('english'))
stemmer=SnowballStemmer('english')
seed=5
```

```
['quora-insincere-questions-classification']
```

```
In [2]: from keras.models import Sequential
        from keras.layers import Dense,Embedding,SpatialDropout1D,Dropout,CuDNNLSTM,Bidirectio
        from keras.utils import to_categorical
        from keras import backend
        from keras.preprocessing.text import Tokenizer
        from keras.preprocessing.sequence import pad_sequences
        from keras.callbacks import Callback,EarlyStopping,ReduceLROnPlateau
```

```
Using TensorFlow backend.
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:516: FutureWarning

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:517: FutureWarning

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:518: FutureWarning

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:519: FutureWarning

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:520: FutureWarning

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: Future

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: Future

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: Future

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,
```

```
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: Future

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:545: Future

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,

/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: Future

Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy,
```

```
In [3]: data=pd.read_csv('../input/quora-insincere-questions-classification/train.csv')
        test=pd.read_csv('../input/quora-insincere-questions-classification/test.csv')
        data.head()
```

Out[3]:

| | qid | question_text | target |
|---|---|---|---|
| 0 | 00002165364db923c7e6 | How did Quebec nationalists see their province... | 0 |
| 1 | 000032939017120e6e44 | Do you have an adopted dog, how would you enco... | 0 |
| 2 | 0000412ca6e4628ce2cf | Why does velocity affect time? Does velocity a... | 0 |
| 3 | 000042bf85aa498cd78e | How did Otto von Guericke used the Magdeburg h... | 0 |
| 4 | 0000455dfa3e01eae3af | Can I convert montra helicon D to a mountain b... | 0 |

```
In [4]: #target1=data[data['target']==1]
        #target0=data[data['target']==0]
        #sampled_size=target1.shape[0]
        #sampled_target0=target0.sample(sampled_size,random_state=seed)
        #new_data=pd.concat([target1,sampled_target0],axis=0)
        #Shuffling the data
        #new_data=new_data.sample(frac=1,random_state=seed).reset_index(drop=True)
```

```
In [5]: new_data=data
```

pre-processing for deeplearning, Removing only the punctuations. Stop words might be helpful.

```
In [6]: def filter_for_nn(text):
            tokenized_word=word_tokenize(text)
            filtered_words=[word.lower() for word in tokenized_word if word not in string.punct
            return ' '.join(filtered_words)
```

```
In [7]: new_data['question_text']=new_data['question_text'].apply(lambda x: filter_for_nn(x))

In [8]: X=new_data.question_text
        Y=new_data.target

        train_X,val_X,train_y,val_y=train_test_split(X,Y,test_size=0.01,random_state=seed)
```

Converting text to sequences

```
In [9]: token=Tokenizer()
        token.fit_on_texts(train_X.values)
        train_seq=token.texts_to_sequences(train_X.values)
        val_seq=token.texts_to_sequences(val_X.values)
```

Padding the sequences

```
In [10]: maxlen=100
         train_pad=pad_sequences(train_seq,maxlen=maxlen)
         val_pad=pad_sequences(val_seq,maxlen=maxlen)

In [11]: train_y.shape

Out[11]: (1293060,)

In [12]: train_y=to_categorical(train_y.values)
         val_y=to_categorical(val_y.values)

In [13]: vocabulary=token.word_index

In [14]: train_y.shape

Out[14]: (1293060, 2)
```

Getting our own Embeddings.

```
In [15]: embedding_len=300
         model=Sequential()
         model.add(Embedding(len(vocabulary)+1,embedding_len,input_length=maxlen))
         model.add(SpatialDropout1D(0.3))
         model.add(Bidirectional(CuDNNLSTM(100)))
         model.add(Dense(512,activation='relu'))
         model.add(Dropout(0.5,seed=seed))
         #model.add(Dense(512,activation='relu'))
         #model.add(Dropout(0.5,seed=seed))
         model.add(Dense(2,activation='softmax'))
         model.compile(loss='binary_crossentropy',optimizer='adam')

         print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 300)          60564600
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 100, 300)          0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 200)               321600
_________________________________________________________________
dense_1 (Dense)              (None, 512)               102912
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
=================================================================
Total params: 60,990,138
Trainable params: 60,990,138
Non-trainable params: 0
_________________________________________________________________
None
```

Using Callbacks.

```
In [16]: est=EarlyStopping(monitor='val_loss',patience=10,restore_best_weights=True)
         rlr=ReduceLROnPlateau(monitor='val_loss',patience=10,min_lr=0.0001,factor=0.1)
         call_back=[est,rlr]
```

```
In [17]: result1=model.fit(train_pad,train_y,epochs=2,batch_size=512,validation_data=[val_pad,v
```

```
Train on 1293060 samples, validate on 13062 samples
Epoch 1/2
1293060/1293060 [==============================] - 168s 130us/step - loss: 0.1185 - val_loss: 0
Epoch 2/2
1293060/1293060 [==============================] - 158s 123us/step - loss: 0.0970 - val_loss: 0
```

```
In [18]: pred1_proba=model.predict(val_pad)
         pred1=np.argmax(pred1_proba,axis=1)
         score1=f1_score(np.argmax(val_y,axis=1),pred1)
         print("F1 score for Validation data without pre-trained embeddings:",score1)

         train_pred1_proba=model.predict(train_pad)
         train_pred1=np.argmax(train_pred1_proba,axis=1)
         train_score1=f1_score(np.argmax(train_y,axis=1),train_pred1)
         print("F1 score for Train data without pre-trained embeddings:",train_score1)
```

```
F1 score for Validation data without pre-trained embeddings: 0.6124469589816125
F1 score for Train data without pre-trained embeddings: 0.7150980119560684
```

Simple Bidirectional LSTM with own embeddings with only 5 epochs has worked better than XGBClassifier! But this model used significantly more data than XGBClassifier. But we can see that this model is overfitting. You can train it for around 100 epochs. We have set patience=10 in callbacks and so they are definitely not effective in 5 epochs. During 100 epochs, they will help in not overfitting. You can reduce the learning rate of Adam optimizer to reduce overfitting aswell.

Taking Embeddings from GloVe

```
In [19]: Embedding_file='../input/quora-insincere-questions-classification/embeddings/glove.840
         def get_coefs(word,*arr):return word,np.asarray(arr,dtype='float32')
         embeddings_index=dict(get_coefs(*o.split(" ")) for o in open(Embedding_file))

In [20]: embedding_stack=np.stack(embeddings_index.values())
         emb_mean,emb_std=embedding_stack.mean(),embedding_stack.std()
         embedding_matrix=np.random.normal(emb_mean,emb_std,(len(vocabulary)+1,embedding_stack
         for word,i in vocabulary.items():
             embedding_vector=embeddings_index.get(word)
             if embedding_vector is not None:
                 embedding_matrix[i]=embedding_vector

/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3249: FutureWarning:

arrays to stack must be passed as a "sequence" type such as list or tuple. Support for non-sequ



In [21]: model2=Sequential()
         model2.add(Embedding(embedding_matrix.shape[0],embedding_matrix.shape[1],weights=[embe
         model2.add(SpatialDropout1D(0.3))
         model2.add(Bidirectional(CuDNNLSTM(100)))
         model2.add(Dense(512,activation='relu'))
         model2.add(Dropout(0.5,seed=seed))
         #model2.add(Dense(512,activation='relu'))
         #model2.add(Dropout(0.8,seed=seed))
         model2.add(Dense(2,activation='softmax'))
         model2.compile(loss='binary_crossentropy',optimizer='adam')

In [22]: result2=model2.fit(train_pad,train_y,epochs=2,batch_size=64,validation_data=[val_pad,v

Train on 1293060 samples, validate on 13062 samples
Epoch 1/2
1293060/1293060 [==============================] - 682s 528us/step - loss: 0.1106 - val_loss: 0
Epoch 2/2
 186560/1293060 [===>...] - ETA: 9:40 - loss: 0.0931

In [23]: pred2_proba=model2.predict(val_pad)
         pred2=np.argmax(pred2_proba,axis=1)
         score2=f1_score(np.argmax(val_y,axis=1),pred2)
         print("F1 score for Validation data using GloVe Embeddings:",score2)
```

```
train_pred2_proba=model2.predict(train_pad)
train_pred2=np.argmax(train_pred2_proba,axis=1)
train_score2=f1_score(np.argmax(train_y,axis=1),train_pred2)
print("F1 Score for Train data using GloVe Embeddings:",train_score2)
```

```
F1 score for Validation data using GloVe Embeddings: 0.6657700067249496
F1 Score for Train data using GloVe Embeddings: 0.7331004401286835
```

Using GloVe Embeddings on the same model has increased the score significantly. Here again, the model is overfitting. Thus, we see that neural networks are extremely powerful.

In [ ]: