```
# coding: utf-8
### Orginal Notebook Created by CIEP / Global DDM COE
#### Nidhi Sawhney, Stojan Maleschlijski & Ian Henry

# In[16]:

import numpy as np  # you probably don't need this line
from glob import glob
import os
import sys
import io
import pyhdb

from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import f1_score


# In[27]:

from keras.preprocessing import sequence
from keras.utils import np_utils

from keras.models import Sequential
from keras.layers import Embedding
from keras.layers.core import Flatten,Dense,Dropout
from keras.optimizers import SGD,Adam
from keras.layers.convolutional import *

from keras.layers import SpatialDropout1D
from keras.layers import Merge,Input
from keras.models import Model

import base64
hanauid='My User
hanapw='My Pass'


# In[29]:

from imblearn.over_sampling import SMOTE
from sklearn.metrics import confusion_matrix
import sklearn
print(sklearn.__version__)


# In[4]:

import re
from numpy.random import normal
import gensim.models.keyedvectors as Word2vec


# In[5]:

get_ipython().magic('matplotlib inline')


# In[6]:

def createBinaryModel(input_length, vocab_size) :
    model = Sequential([
    Embedding(vocab_size, 32, input_length=input_length,
              dropout=0.2),
    SpatialDropout1D(0.2),
    Dropout(0.25),
    Convolution1D(64, 5, padding='same', activation='relu'),
    Dropout(0.25),
    MaxPooling1D(),
    Flatten(),
```

```
    Dense(100, activation='relu'),
    Dropout(0.7),
    Dense(1, activation='sigmoid')])
    model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
    model.summary()
    return model


# In[7]:

def createMultiClassModel(num_labels,input_length , vocab_size) :
    model = Sequential([
    Embedding(vocab_size, 32, input_length=input_length,
              dropout=0.2),
    SpatialDropout1D(0.2),
    Dropout(0.25),
    Convolution1D(64, 5, padding='same', activation='relu'),
    Dropout(0.25),
    MaxPooling1D(),
    Flatten(),
    Dense(100, activation='relu'),
    Dropout(0.7),
    Dense(num_labels, activation='softmax')])
    model.compile(loss='categorical_crossentropy', optimizer=Adam(),
    metrics=['accuracy'])
    model.summary()
    return model


# In[8]:

def createMultiClassModelWithPTV(num_labels, input_length, vocab_size, emb) :
    model = Sequential([
    Embedding(vocab_size, len(emb[0]), input_length=input_length,
              weights=[emb], trainable=False),
    SpatialDropout1D(0.2),
    Dropout(0.25),
    Convolution1D(64, 5, padding='same', activation='relu'),
    Dropout(0.25),
    MaxPooling1D(),
    Flatten(),
    Dense(100, activation='relu'),
    Dropout(0.7),
    Dense(num_labels, activation='softmax')])
    model.compile(loss='categorical_crossentropy', optimizer=Adam(),
    metrics=['accuracy'])
    model.summary()
    return model


# In[26]:

cnxn = pyhdb.connect(
    host="mo-3fda111e5.mo.sap.corp",
    port=30041,
    user=hanauid,
    password=hanapw
)


# In[28]:

def getTrainingData() :

    cursor = cnxn.cursor()
    cursor.execute("CALL FAKENEWS.GET_REVIEW_DATA()")
    data = cursor.fetchall()
    data_array=np.asarray(data)
    token_id = ((data_array[:,3]).astype(int))
```

```
    data_ids = ((data_array[:,[0,3]]).astype(int))
    file_names = (data_array[:,4])
    return data_ids


# In[11]:

def formatTrainingData(raw_training_data, vocab_size, seq_len) :
    document_ids = list(set(raw_training_data[:,0]))
    xval = []
    l=(raw_training_data[:,0])
    for i in document_ids :
        #sublist = list(data_ids[(get_indexes(i,l)),1])
        sublist = (raw_training_data[np.where(l == i),1]).tolist()[0]
        xval.append(sublist)

    print(len(xval))
    #Make rare words equal
    val = [np.array([i if i<vocab_size-1 else vocab_size-1 for i in s]) for s in xval]
    #Make matrix of same size by padding zeros or truncating
    val = sequence.pad_sequences(val, maxlen=seq_len, value=0)

    print(val.shape)
    return val




# In[12]:

def getLabels() :

    cursor = cnxn.cursor()

    cursor.execute("SELECT FAKE FROM FAKENEWS.TITLE_140 ORDER BY ID")
    labels = cursor.fetchall()
    labels_array=np.asarray(labels)
    labels = ((labels_array[:,0]).astype(int))

    sub_labels = labels_array[np.where(labels_array > 0)]
    return sub_labels


# In[13]:

def getBinaryLabels() :

    cursor = cnxn.cursor()

    cursor.execute("SELECT FAKE FROM FAKENEWS.TITLE_140 ORDER BY ID")
    labels = cursor.fetchall()
    labels_array=np.asarray(labels)
    labels = ((labels_array[:,0]).astype(int))

    return labels


# In[21]:

def getWords() :

    cursor = cnxn.cursor()
    cursor.execute("CALL FAKENEWS.GET_WORDS()")
    data = cursor.fetchall()
    words_array=np.asarray(data)
    print(len(words_array))
    return words_array


```

```
# In[15]:


def create_emb(words,wikimodel, vocab_size, n_fact):
    #n_fact = model.shape[1]
    #n_fact = 300
    emb = np.zeros((vocab_size, n_fact))

    for i in range(1,len(emb)):
        word = words_array[i,1] #wikimodel.wv.index2word[i]
        if word and re.match(r"^[a-zA-Z\-]*$", word):
            #src_idx = wordidx[word]
            #src_idx = wikimodel.vocab[word].index
            #emb[i] = vecs[src_idx]
            try:
                emb[i] = wikimodel.wv[word.lower()]

            except:
                 emb[i] = normal(scale=0.6, size=(n_fact,))
        else:
            # If we can't find the word in glove, randomly initialize
            emb[i] = normal(scale=0.6, size=(n_fact,))

    # This is our "rare word" id - we want to randomly initialize
    emb[-1] = normal(scale=0.6, size=(n_fact,))
    emb/=3
    return emb


# In[16]:

def createCustomEmbedding(wordVector) :

    wikimodel = Word2vec.KeyedVectors.load_word2vec_format(wordVector, binary=False)
    emb=create_emb(getWords(), wikimodel,500,300)
    return emb


# In[17]:

test = getBinaryLabels()
sum(test)


# In[ ]:
```