สมาชิก

นายณัฐสิทธิ์ หมานละงู 6410210104

นายฟาร์ดี เกปัน 6410210235

นายสันเพ็ชร แซ่ฟัง 6410210319

Link: https://www.kaggle.com/nattasitnts/lstm-v2/edit?fbclid=IwAR0GoZ6wb63c2MCpefBUWBzzVGgawFyr1w17bvGnsyioldX05PWo9yZtqQ0

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = ':https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F16%2F799881%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-161607.iam.gs

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
```

```
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

Created by Peter Nagy February 2017 **

Github

Linkedin

****Sentiment Analysis:** the process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral.

สมาชิก นายณัฐสิทธิ์ หมานละงู 6410210 นายฟาร์ดี เกบัน นายสันเพ็ชร แซ่ฟัง

As an improvement to my previous Kernel, here I am trying to achieve better results with a Recurrent Neural Network.
You may want to check out my latest kernel on an LSTM multi-class classification problem.

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

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 sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re
from sklearn.utils import shuffle
from sklearn.utils import resample
from sklearn.metrics import confusion_matrix,classification_report
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory
```

Only keeping the necessary columns.

```
data = pd.read_csv('../input/Sentiment.csv')
# Keeping only the neccessary columns
data = data[['text','sentiment']]
```

Next, I am dropping the 'Neutral' sentiments as my goal was to only differentiate positive and negative tweets. After that, I am filtering the tweets so only valid texts and words remain. Then, I define the number of max features as 2000 and use Tokenizer to vectorize and convert text into Sequences so the Network can deal with it as input.

```
data = data[data.sentiment != "Neutral"]
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))

print(data[ data['sentiment'] == 'Positive'].size)
print(data[ data['sentiment'] == 'Negative'].size)

for idx,row in data.iterrows():
    row[0] = row[0].replace('rt',' ')

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)
```

```
4472
16986
```

```
# Separate majority and minority classes
data_majority = data[data['sentiment'] == 'Negative']
data_minority = data[data['sentiment'] == 'Positive']

bias = data_minority.shape[0]/data_majority.shape[0]
# lets split train/test data first then
train = pd.concat([data_majority.sample(frac=0.8,random_state=200),
         data_minority.sample(frac=0.8,random_state=200)])
test = pd.concat([data_majority.drop(data_majority.sample(frac=0.8,random_state=200).index),
        data_minority.drop(data_minority.sample(frac=0.8,random_state=200).index)])

train = shuffle(train)
test = shuffle(test)
```

```
print('positive data in training:',(train.sentiment == 'Positive').sum())
print('negative data in training:',(train.sentiment == 'Negative').sum())
print('positive data in test:',(test.sentiment == 'Positive').sum())
print('negative data in test:',(test.sentiment == 'Negative').sum())
```

```
positive data in training: 1789
negative data in training: 6794
positive data in test: 447
negative data in test: 1699
```

```
# Separate majority and minority classes in training data for upsampling
data_majority = train[train['sentiment'] == 'Negative']
data_minority = train[train['sentiment'] == 'Positive']

print("majority class before upsample:",data_majority.shape)
print("minority class before upsample:",data_minority.shape)

# Upsample minority class
data_minority_upsampled = resample(data_minority,
                                   replace=True,     # sample with replacement
                                   n_samples= data_majority.shape[0],    # to match majority class
                                   random_state=123) # reproducible results

# Combine majority class with upsampled minority class
data_upsampled = pd.concat([data_majority, data_minority_upsampled])

# Display new class counts
print("After upsampling\n",data_upsampled.sentiment.value_counts(),sep = "")

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values) # training with whole data

X_train = tokenizer.texts_to_sequences(data_upsampled['text'].values)
X_train = pad_sequences(X_train,maxlen=29)
Y_train = pd.get_dummies(data_upsampled['sentiment']).values
print('x_train shape:',X_train.shape)

X_test = tokenizer.texts_to_sequences(test['text'].values)
X_test = pad_sequences(X_test,maxlen=29)
Y_test = pd.get_dummies(test['sentiment']).values
print("x_test shape", X_test.shape)
```

```
majority class before upsample: (6794, 2)
minority class before upsample: (1789, 2)
After upsampling
Negative    6794
Positive    6794
Name: sentiment, dtype: int64
x_train shape: (13588, 29)
x_test shape (2146, 29)
```

Next, I compose the LSTM Network. Note that **embed_dim**, **lstm_out**, **batch_size**, **droupout_x** variables are hyperparameters, their values are somehow intuitive, can be and must be played with in order to achieve good results. Please also note that I am using softmax as activation function. The reason is that our Network is using categorical crossentropy, and softmax is just the right activation method for that.

```
embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, 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_7 (Embedding)      (None, 28, 128)           256000
```

```
spatial_dropout1d_7 (Spatial (None, 28, 128)           0         
_________________________________________________________________
lstm_7 (LSTM)                (None, 196)               254800    
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 394       
=================================================================
Total params: 511,194
Trainable params: 511,194
Non-trainable params: 0
_________________________________________________________________
None
```

Hereby I declare the train and test dataset.

```
Y = pd.get_dummies(data['sentiment']).values
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.33, random_state = 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

```
(7188, 28) (7188, 2)
(3541, 28) (3541, 2)
```

Here we train the Network. We should run much more than 7 epoch, but I would have to wait forever for kaggle, so it is 7 for now.

```
batch_size = 128
# also adding weights

model.fit(X_train, Y_train, epochs = 10, batch_size=batch_size, verbose = 1,
          class_weight=class_weights)
```

```
Epoch 1/10
7188/7188 [==============================] - 14s 2ms/step - loss: 1.2649 - acc: 0.3706
Epoch 2/10
7188/7188 [==============================] - 11s 2ms/step - loss: 0.9040 - acc: 0.7298
Epoch 3/10
7188/7188 [==============================] - 11s 2ms/step - loss: 0.7424 - acc: 0.7859
Epoch 4/10
7188/7188 [==============================] - 12s 2ms/step - loss: 0.6648 - acc: 0.8141
Epoch 5/10
7188/7188 [==============================] - 11s 2ms/step - loss: 0.6256 - acc: 0.8253
Epoch 6/10
7188/7188 [==============================] - 11s 2ms/step - loss: 0.5625 - acc: 0.8452
Epoch 7/10
7188/7188 [==============================] - 12s 2ms/step - loss: 0.5053 - acc: 0.8630
Epoch 8/10
7188/7188 [==============================] - 11s 2ms/step - loss: 0.4842 - acc: 0.8646
Epoch 9/10
7188/7188 [==============================] - 11s 2ms/step - loss: 0.4472 - acc: 0.8798
Epoch 10/10
7188/7188 [==============================] - 12s 2ms/step - loss: 0.4134 - acc: 0.8932
<keras.callbacks.History at 0x7a39e4b99470>
```

```
Y_pred = model.predict_classes(X_test,batch_size = batch_size)
df_test = pd.DataFrame({'true': Y_test.tolist(), 'pred':Y_pred})
df_test['true'] = df_test['true'].apply(lambda x: np.argmax(x))
print("confusion matrix",confusion_matrix(df_test.true, df_test.pred))
print(classification_report(df_test.true, df_test.pred))
```

```
confusion matrix [[2214  607]
 [ 169  551]]
             precision    recall  f1-score   support
```

```
             0       0.93      0.78      0.85      2821
             1       0.48      0.77      0.59       720

   avg / total       0.84      0.78      0.80      3541
```

Extracting a validation set, and measuring score and accuracy.

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

```
   score: 0.58
   acc: 0.78
```

Finally measuring the number of correct guesses. It is clear that finding negative tweets goes very well for the Network but deciding whether is positive is not really. My educated guess here is that the positive training set is dramatically smaller than the negative, hence the "bad" results for positive tweets.

```
pos_cnt, neg_cnt, pos_correct, neg_correct = 0, 0, 0, 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:
            neg_correct += 1
        else:
            pos_correct += 1

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



print("pos_acc", pos_correct/pos_cnt*100, "%")
print("neg_acc", neg_correct/neg_cnt*100, "%")
```

```
   pos_acc 77.34627831715211 %
   neg_acc 78.50545759865659 %
```

```
Y_pred = model.predict_classes(X_test,batch_size = batch_size)
df_test = pd.DataFrame({'true': Y_test.tolist(), 'pred':Y_pred})
df_test['true'] = df_test['true'].apply(lambda x: np.argmax(x))
print("confusion matrix",confusion_matrix(df_test.true, df_test.pred))
print(classification_report(df_test.true, df_test.pred))
```

```
   confusion matrix [[1279  351]
    [  99  312]]
                precision    recall  f1-score   support
```

```
             0       0.93      0.78      0.85      1630
             1       0.47      0.76      0.58       411

   avg / total       0.84      0.78      0.80      2041
```

```
twt = ["Loved the book! Couldn't put it down!"]
#vectorizing the tweet by the pre-fitted tokenizer instance
twt = tokenizer.texts_to_sequences(twt)
#padding the tweet to have exactly the same shape as `embedding_2` input
twt = pad_sequences(twt, maxlen=28, dtype='int32', value=0)
print(twt)
sentiment = model.predict(twt,batch_size=1,verbose = 2)[0]
if(np.argmax(sentiment) == 0):
    print("negative")
elif (np.argmax(sentiment) == 1):
    print("positive")
```

```
   [[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0  528    1 1347  435   13  313]]
    positive
```

```
twt = ["Terrible experience at the store today. Avoid!"]
#vectorizing the tweet by the pre-fitted tokenizer instance
twt = tokenizer.texts_to_sequences(twt)
#padding the tweet to have exactly the same shape as `embedding_2` input
twt = pad_sequences(twt, maxlen=28, dtype='int32', value=0)
print(twt)
sentiment = model.predict(twt,batch_size=1,verbose = 2)[0]
if(np.argmax(sentiment) == 0):
    print("negative")
elif (np.argmax(sentiment) == 1):
    print("positive")
```

```
   [[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
        0    0    0    0    0    0    0    0    0 1003   29    1  460 1775]]
    negative
```

```
twt = ["Best coffee in town! Can't start my day without it."]
#vectorizing the tweet by the pre-fitted tokenizer instance
twt = tokenizer.texts_to_sequences(twt)
#padding the tweet to have exactly the same shape as `embedding_2` input
twt = pad_sequences(twt, maxlen=28, dtype='int32', value=0)
print(twt)
sentiment = model.predict(twt,batch_size=1,verbose = 2)[0]
if(np.argmax(sentiment) == 0):
    print("negative")
elif (np.argmax(sentiment) == 1):
    print("positive")
```

```
   [[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
       0   0   0   0 174  12  53 410 568  13]]
    positive
```

```
twt = ["Absolutely stunning sunset! Nature at its finest."]
#vectorizing the tweet by the pre-fitted tokenizer instance
twt = tokenizer.texts_to_sequences(twt)
#padding the tweet to have exactly the same shape as `embedding_2` input
twt = pad_sequences(twt, maxlen=28, dtype='int32', value=0)
print(twt)
sentiment = model.predict(twt,batch_size=1,verbose = 2)[0]
if(np.argmax(sentiment) == 0):
    print("negative")
elif (np.argmax(sentiment) == 1):
    print("positive")
```

```
[[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0  646   29   81 1892]]
positive
```

```
twt = ["Worst customer service ever. Frustrating and unhelpful."]
#vectorizing the tweet by the pre-fitted tokenizer instance
twt = tokenizer.texts_to_sequences(twt)
#padding the tweet to have exactly the same shape as `embedding_2` input
twt = pad_sequences(twt, maxlen=28, dtype='int32', value=0)
print(twt)
sentiment = model.predict(twt,batch_size=1,verbose = 2)[0]
if(np.argmax(sentiment) == 0):
    print("negative")
elif (np.argmax(sentiment) == 1):
    print("positive")
```

```
[[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0  773 1699  307    8]]
negative
```