Configuration

In [1]:

```
# Parameters
PROJECT_NAME = 'ML1010-Group-Project'
ENABLE_COLAB = False

#Root Machine Learning Directory. Projects appear underneath
GOOGLE_DRIVE_MOUNT = '/content/gdrive'
COLAB_ROOT_DIR = GOOGLE_DRIVE_MOUNT + '/MyDrive/Colab Notebooks'
COLAB_INIT_DIR = COLAB_ROOT_DIR + '/utility_files'

LOCAL_ROOT_DIR = '/home/magni//ML_Root/project_root'
LOCAL_INIT_DIR = LOCAL_ROOT_DIR + '/utility_files'
```

Bootstrap Environment

In [2]:

```
#add in support for utility file directory and importing
import sys
import os

if ENABLE_COLAB:
  #Need access to drive
  from google.colab import drive
  drive.mount(GOOGLE_DRIVE_MOUNT, force_remount=True)

  #add in utility directory to syspath to import
  INIT_DIR = COLAB_INIT_DIR
  sys.path.append(os.path.abspath(INIT_DIR))

  #Config environment variables
  ROOT_DIR = COLAB_ROOT_DIR

else:
  #add in utility directory to syspath to import
  INIT_DIR = LOCAL_INIT_DIR
  sys.path.append(os.path.abspath(INIT_DIR))

  #Config environment variables
  ROOT_DIR = LOCAL_ROOT_DIR

#Import Utility Support
from jarvis import Jarvis
jarvis = Jarvis(ROOT_DIR, PROJECT_NAME)

import mv_python_utils as mvutils
```

```
Wha...where am I?
I am awake now.

I have set your current working directory to /home/magni/ML_Root/project_root
/ML1010-Group-Project
The current time is 19:14
```

```
Hello sir. I hope you had dinner.
```

Setup Runtime Environment

In [3]:

```
if ENABLE_COLAB:
  #!pip install scipy -q
  #!pip install scikit-learn -q
  #!pip install pycaret -q
  #!pip install matplotlib -q
  #!pip install joblib -q
  #!pip install pandasql -q

  display('Google Colab enabled')
else:
  display('Google Colab not enabled')

#Common imports
import json
import gzip
import pandas as pd
import numpy as np
import matplotlib
import re
import nltk
import matplotlib.pyplot as plt

pd.set_option('mode.chained_assignment', None)
nltk.download('stopwords')
%matplotlib inline
```

```
'Google Colab not enabled'
```

```
[nltk_data] Downloading package stopwords to /home/magni/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Load Data

In [5]:

```
# From article https://machinelearningmastery.com/sequence-classification-lst

import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence

# fix random seed for reproducibility
numpy.random.seed(7)
```

In [6]:

```
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
```

In [27]:

```
print(X_train.shape)
#print(X_train[1])
#print(type(X_train))
print(y_test[4])
#print(X_train)
tDf = pd.DataFrame(X_train)
tDf.head()
```

```
(25000, 500)
1
```

Out[27]:

	0	1	2	3	4	5	6	7	8	9	...	490	491	492	493	494	495	496	497	498	499
0	0	0	0	0	0	0	0	0	0	0	...	4472	113	103	32	15	16	2	19	178	32
1	0	0	0	0	0	0	0	0	0	0	...	52	154	462	33	89	78	285	16	145	95
2	0	0	0	0	0	0	0	0	0	0	...	106	607	624	35	534	6	227	7	129	113
3	687	23	4	2	2	6	3693	42	38	39	...	26	49	2	15	566	30	579	21	64	2574
4	0	0	0	0	0	0	0	0	0	0	...	19	14	5	2	6	226	251	7	61	113

5 rows × 500 columns

In [21]:

```
# truncate and pad input sequences
max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
```

In [22]:

```
print(X_train.shape)
print(X_train[9])
# print(type(X_train))
```

```
(25000, 500)
[    0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    1   14   20   47  111  439 3445   19
   12   15  166   12  216  125   40    6  364  352  707 1187   39  294
   11   22  396   13   28    8  202   12 1109   23   94    2  151  111
  211  469    4   20   13  258  546 1104    2   12   16   38   78   33
  211   15   12   16 2849   63   93   12    6  253  106   10   10   48
  335  267   18    6  364 1242 1179   20   19    6 1009    7 1987  189
    5    6    2    7 2723    2   95 1719    6    2    7 3912    2   49
  369  120    5   28   49  253   10   10   13 1041   19   85  795   15
    4  481    9   55   78  807    9  375    8 1167    8  794   76    7
```

In [8]:

```
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(x=top_words,
                    y=embedding_vecor_length,
                    input_length=max_review_length))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
print(model.summary())
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3, batch
```

```
2022-01-11 09:33:18.673190: W tensorflow/stream_executor/platform/default/dso
_loader.cc:64] Could not load dynamic library 'libcuda.so.1'; dlerror: libcud
a.so.1: cannot open shared object file: No such file or directory
2022-01-11 09:33:18.673229: W tensorflow/stream_executor/cuda/cuda_driver.cc:
269] failed call to cuInit: UNKNOWN ERROR (303)
2022-01-11 09:33:18.673260: I tensorflow/stream_executor/cuda/cuda_diagnostic
s.cc:156] kernel driver does not appear to be running on this host (localhos
t.localdomain): /proc/driver/nvidia/version does not exist
2022-01-11 09:33:18.673532: I tensorflow/core/platform/cpu_feature_guard.cc:1
51] This TensorFlow binary is optimized with oneAPI Deep Neural Network Libra
ry (oneDNN) to use the following CPU instructions in performance-critical ope
rations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate c
ompiler flags.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 500, 32)           160000

 lstm (LSTM)                 (None, 100)               53200

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/3
391/391 [==============================] - 134s 338ms/step - loss: 0.4235 - a
ccuracy: 0.7976 - val_loss: 0.3114 - val_accuracy: 0.8717
Epoch 2/3
391/391 [==============================] - 133s 341ms/step - loss: 0.2684 - a
ccuracy: 0.8948 - val_loss: 0.3634 - val_accuracy: 0.8666
Epoch 3/3
391/391 [==============================] - 133s 340ms/step - loss: 0.2342 - a
```

Out[8]:

```
<keras.callbacks.History at 0x7fe6807be0d0>
```

In [9]:

```
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 87.98%
```

In [11]:

```
# Same as above but with dropout layers added

# LSTM with Dropout for sequence classification in the IMDB dataset

# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
# truncate and pad input sequences
max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_revie
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
print(model.summary())
model.fit(X_train, y_train, epochs=3, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 500, 32)           160000    
                                                                 
 dropout (Dropout)           (None, 500, 32)           0         
 lstm_1 (LSTM)               (None, 100)               53200

 dropout_1 (Dropout)         (None, 100)               0

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/3
391/391 [==============================] - 104s 262ms/step - loss: 0.4718 - a
ccuracy: 0.7743
Epoch 2/3
391/391 [==============================] - 103s 264ms/step - loss: 0.3608 - a
ccuracy: 0.8517
Epoch 3/3
391/391 [==============================] - 103s 264ms/step - loss: 0.2664 - a
ccuracy: 0.8952
Accuracy: 86.28%
```

In [12]:

```
# LSTM with dropout for sequence classification in the IMDB dataset
# Added a specific dropout on the LSTM layer instead of separate layer

# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
# truncate and pad input sequences
max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_revie
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
print(model.summary())
model.fit(X_train, y_train, epochs=3, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 500, 32)           160000

 lstm_2 (LSTM)               (None, 100)               53200

 dense_2 (Dense)             (None, 1)                 101
```

```
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/3
391/391 [==============================] - 144s 363ms/step - loss: 0.4658 - a
ccuracy: 0.7733
Epoch 2/3
391/391 [==============================] - 142s 363ms/step - loss: 0.3229 - a
ccuracy: 0.8682
Epoch 3/3
391/391 [==============================] - 141s 361ms/step - loss: 0.2758 - a
ccuracy: 0.8855
Accuracy: 86.51%
```

In [13]:

```
#We can easily add a one-dimensional CNN and max pooling layers after
#the Embedding layer which then feed the consolidated features to the LSTM.
#We can use a smallish set of 32 features with a small filter
#length of 3. The pooling layer can use the standard length of 2
#to halve the feature map size.

# LSTM and CNN for sequence classification in the IMDB dataset
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
# truncate and pad input sequences
max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_revie
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
print(model.summary())
model.fit(X_train, y_train, epochs=3, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_4"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_4 (Embedding)     (None, 500, 32)           160000    
                                                                 
 conv1d (Conv1D)             (None, 500, 32)           3104      
                                                                 
 max_pooling1d (MaxPooling1D  (None, 250, 32)          0         
 )                                                               
                                                                 
 lstm_3 (LSTM)               (None, 100)               53200     
                                                                 
 dense_3 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 216,405
Trainable params: 216,405
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/3
391/391 [==============================] - 56s 139ms/step - loss: 0.4232 - ac
curacy: 0.7901
Epoch 2/3
391/391 [==============================] - 55s 142ms/step - loss: 0.2470 - ac
curacy: 0.9028
Epoch 3/3
391/391 [==============================] - 56s 142ms/step - loss: 0.1999 - ac
curacy: 0.9235
```

In [14]:

```
#Same as above but added additional epochs

# LSTM and CNN for sequence classification in the IMDB dataset
import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
# truncate and pad input sequences
max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_revie
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
print(model.summary())
model.fit(X_train, y_train, epochs=6, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_5 (Embedding)     (None, 500, 32)           160000    
                                                                 
 conv1d_1 (Conv1D)           (None, 500, 32)           3104      
                                                                 
 max_pooling1d_1 (MaxPooling  (None, 250, 32)          0         
 1D)                                                             
                                                                 
 lstm_4 (LSTM)               (None, 100)               53200     
                                                                 
 dense_4 (Dense)             (None, 1)                 101       
                                                                 
=================================================================
Total params: 216,405
Trainable params: 216,405
Non-trainable params: 0
_________________________________________________________________
None
```

```
Epoch 1/6
391/391 [==============================] - 55s 139ms/step - loss: 0.4326 - ac
curacy: 0.7930
Epoch 2/6
391/391 [==============================] - 54s 139ms/step - loss: 0.2471 - ac
curacy: 0.9028
Epoch 3/6
391/391 [==============================] - 55s 140ms/step - loss: 0.2010 - ac
curacy: 0.9246
Epoch 4/6
391/391 [==============================] - 55s 140ms/step - loss: 0.1730 - ac
curacy: 0.9356
Epoch 5/6
391/391 [==============================] - 55s 140ms/step - loss: 0.1421 - ac
curacy: 0.9480
Epoch 6/6
391/391 [==============================] - 55s 140ms/step - loss: 0.1097 - ac
curacy: 0.9633
```