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 13:32
```

```
Hello sir. Reminder, no more coffee.
```

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!
```

Import necessary depencencies

In [4]:

```
import pandas as pd
import numpy as np
import text_normalizer as tn
from tensorflow import keras
#import model_evaluation_utils as meu

np.set_printoptions(precision=2, linewidth=80)
```

```
2022-01-11 13:32:23.061281: W tensorflow/stream_executor/platform/default/dso
_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: l
ibcudart.so.11.0: cannot open shared object file: No such file or directory
2022-01-11 13:32:23.061307: I tensorflow/stream_executor/cuda/cudart_stub.cc:
29] Ignore above cudart dlerror if you do not have a GPU set up on your machi
ne.
```

Load and normalize data

In [5]:

```
dataset = pd.read_csv(r'/home/magni/ML_Root/project_root/data/ML1010_Weekly/m

# take a peek at the data
print(dataset.head())
reviews = np.array(dataset['review'])
sentiments = np.array(dataset['sentiment'])

# build train and test datasets
#train_reviews = reviews[:35000]
#train_sentiments = sentiments[:35000]
#test_reviews = reviews[35000:]
#test_sentiments = sentiments[35000:]

train_reviews = reviews[:5000]
train_sentiments = sentiments[:5000]
test_reviews = reviews[5000:]
test_sentiments = sentiments[5000:]



# normalize datasets
#norm_train_reviews = tn.normalize_corpus(train_reviews)
#norm_test_reviews = tn.normalize_corpus(test_reviews)
norm_train_reviews = train_reviews
norm_test_reviews = test_reviews
```

```
                                              review sentiment
0  not bother think would see movie great supspen...  negative
1  careful one get mitt change way look kung fu f...  positive
2  chili palmer tired movie know want success mus...  negative
3  follow little know 1998 british film make budg...  positive
4  dark angel cross huxley brave new world percys...  positive
```

In [6]:

```
print(train_sentiments)
print(test_sentiments)
```

```
['negative' 'positive' 'negative' ... 'positive' 'negative' 'negative']
['negative' 'negative' 'negative' ... 'negative' 'positive' 'negative']
```

Tokenize train & test datasets

In [7]:

```
from nltk.tokenize import word_tokenize
#word_tokenize(text)

tokenized_train = [word_tokenize(text) for text in norm_train_reviews]
tokenized_test = [word_tokenize(text) for text in norm_test_reviews]
```

In [26]:

```
print(len(tokenized_train))
print(len(tokenized_test))
print(len(norm_test_reviews))
```

```
5000
45000
```

```
45000
```

Build Vocabulary Mapping (word to index)

In [8]:

```
from collections import Counter

# build word to index vocabulary
token_counter = Counter([token for review in tokenized_train for token in rev
vocab_map = {item[0]: index+1 for index, item in enumerate(dict(token_counter
max_index = np.max(list(vocab_map.values()))
vocab_map['PAD_INDEX'] = 0
vocab_map['NOT_FOUND_INDEX'] = max_index+1
vocab_size = len(vocab_map)
# view vocabulary size and part of the vocabulary map
print('Vocabulary Size:', vocab_size)
print('Sample slice of vocabulary map:', dict(list(vocab_map.items())[10:20])
```

```
Vocabulary Size: 32182
Sample slice of vocabulary map: {'boring': 11, 'terribly': 12, 'predictable':
13, 'interesting': 14, 'start': 15, 'middle': 16, 'film': 17, 'little': 18, '
social': 19, 'drama': 20}
```

Encode and Pad datasets & Encode prediction class labels

In [9]:

```
from keras.preprocessing import sequence
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# get max length of train corpus and initialize label encoder
num_classes=2 # positive -> 1, negative -> 0
max_len = np.max([len(review) for review in tokenized_train])

## Train reviews data corpus
# Convert tokenized text reviews to numeric vectors
train_X = [[vocab_map[token] for token in tokenized_review] for tokenized_rev
train_X = sequence.pad_sequences(train_X, maxlen=max_len) # pad
## Train prediction class labels
# Convert text sentiment labels (negative\positive) to binary encodings (0/1)
#train_y = le.fit_transform(train_sentiments)
train_y = le.fit_transform(train_sentiments)



## Test reviews data corpus
# Convert tokenized text reviews to numeric vectors
test_X = [[vocab_map[token] if vocab_map.get(token) else vocab_map['NOT_FOUND
           for token in tokenized_review]
              for tokenized_review in tokenized_test]
test_X = sequence.pad_sequences(test_X, maxlen=max_len)
## Test prediction class labels
# Convert text sentiment labels (negative\positive) to binary encodings (0/1)
test_y = le.transform(test_sentiments)

# view vector shapes
print('Max length of train review vectors:', max_len)
print('Train review vectors shape:', train_X.shape, ' Test review vectors sha
```

```
Max length of train review vectors: 627
Train review vectors shape: (5000, 627)  Test review vectors shape: (45000, 6
27)
```

Build the LSTM Model Architecture

In [10]:

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, Dropout, SpatialDropout1D
from keras.layers import LSTM

EMBEDDING_DIM = 128 # dimension for dense embeddings for each token
LSTM_DIM = 64 # total LSTM units

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=EMBEDDING_DIM, input_len
model.add(SpatialDropout1D(0.2))
model.add(LSTM(LSTM_DIM, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy", optimizer="adam",
              metrics=["accuracy"])
2022-01-11 13:32:41.472754: 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 13:32:41.472792: W tensorflow/stream_executor/cuda/cuda_driver.cc:
269] failed call to cuInit: UNKNOWN ERROR (303)
2022-01-11 13:32:41.472808: 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 13:32:41.473043: 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.
```

In [11]:

```
print(model.summary())
```

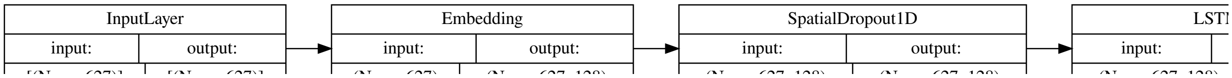
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 627, 128)          4119296   
                                                                 
 spatial_dropout1d (SpatialD  (None, 627, 128)         0         
 ropout1D)                                                       
                                                                 
 lstm (LSTM)                 (None, 64)                49408     
                                                                 
 dense (Dense)               (None, 1)                 65        
                                                                 
=================================================================
Total params: 4,168,769
Trainable params: 4,168,769
Non-trainable params: 0
_________________________________________________________________
None
```

Visualize model architecture

In [12]:

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(model, show_shapes=True, show_layer_names=False,
                 rankdir='LR').create(prog='dot', format='svg'))
```

Out[12]:



Train the model

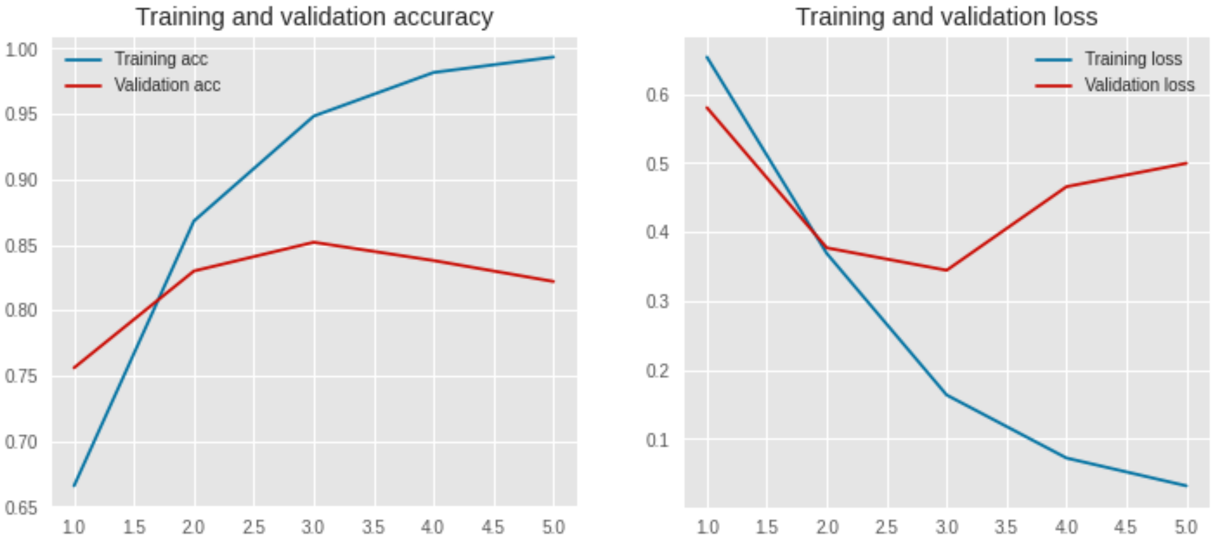
In [13]:

```
batch_size = 100
history = model.fit(train_X,
          train_y,
          epochs=5,
          batch_size=batch_size,
          shuffle=True,
          validation_split=0.1,
          verbose=1)
```

```
Epoch 1/5
45/45 [==============================] - 21s 420ms/step - loss: 0.6536 - accu
racy: 0.6658 - val_loss: 0.5803 - val_accuracy: 0.7560
Epoch 2/5
45/45 [==============================] - 18s 392ms/step - loss: 0.3688 - accu
racy: 0.8682 - val_loss: 0.3769 - val_accuracy: 0.8300
Epoch 3/5
45/45 [==============================] - 18s 407ms/step - loss: 0.1635 - accu
racy: 0.9484 - val_loss: 0.3444 - val_accuracy: 0.8520
Epoch 4/5
45/45 [==============================] - 18s 396ms/step - loss: 0.0721 - accu
racy: 0.9818 - val_loss: 0.4656 - val_accuracy: 0.8380
Epoch 5/5
45/45 [==============================] - 18s 404ms/step - loss: 0.0318 - accu
racy: 0.9936 - val_loss: 0.4996 - val_accuracy: 0.8220
```

In [14]:

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, 'b', label='Training acc')
    plt.plot(x, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, loss, 'b', label='Training loss')
    plt.plot(x, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

plot_history(history)
```



Predict and Evaluate Model Performance

In [16]:

```
pred_test = model.predict(test_X)
```

In [21]:

```
pred_test = np.round(pred_test).astype(int)

pred_test_flat = pred_test.flatten()
print(pred_test_flat)
print(len(pred_test_flat))
print(len(test_X))
```

```
[0 0 1 ... 0 1 0]
45000
45000
```

In [18]:

```
predictions = le.inverse_transform(pred_test.flatten())
```

In [23]:

```
import model_evaluation_utils as meu
meu.display_model_performance_metrics(true_labels=test_sentiments, predicted_
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.8497
Precision: 0.8497
Recall: 0.8497
F1 Score: 0.8497

Model Classification report:
------------------------------
              precision    recall  f1-score   support

    positive       0.85      0.84      0.85     22497
    negative       0.85      0.85      0.85     22503
    accuracy                           0.85     45000
   macro avg       0.85      0.85      0.85     45000
weighted avg       0.85      0.85      0.85     45000


Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive      19000     3497
        negative       3266    19237
```

Introduction

In this project, I classify Yelp round-10 review datasets. The reviews contain a lot of metadata that can be mined and used to infer meaning, business attributes, and sentiment. For simplicity, I classify the review comments into two class: either as positive or negative. Reviews that have star higher than three are regarded as positive while the reviews with star less than or equal to 3 are negative. Therefore, the problem is a supervised learning. To build and train the model, I first tokenize the text and convert them to sequences. Each review comment is limited to 50 words. As a result, short texts less than 50 words are padded with zeros, and long ones are truncated. After processing the review comments, I trained three model in three different ways:

Model-1: In this model, a neural network with LSTM and a single embedding layer were used.

Model-2: In Model-1, an extra 1D convolutional layer has been added on top of LSTM layer to reduce the training time.

Model-3: In this model, I use the same network architecture as Model-2, but use the pre-trained glove 100 dimension word embeddings as initial input.

Since there are about 1.6 million input comments, it takes a while to train the models. To reduce the training time step, I limit the training epoch to three. After three epochs, it is evident that Model-2 is better regarding both training time and validation accuracy.

Project Outline

In this project I will cover the follwouings :

Download data from yelp and process them

Build neural network with LSTM

Build neural network with LSTM and CNN

Use pre-trained GloVe word embeddings

Word Embeddings from Word2Vec

Configuration

In [1]:

```
# Parameters
PROJECT_NAME = 'ML1010_Weekly'
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_Weekly
The current time is 17:10
Hello sir. Reminder, no more coffee.
```

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!
```

Import libraries

In [4]:

```
# Keras
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Flatten, LSTM, Conv1D, MaxPooling1D, Dropout,
from keras.layers.embeddings import Embedding

## Plot
import plotly.offline as py
import plotly.graph_objs as go
py.init_notebook_mode(connected=True)
import matplotlib as plt

# NLTK
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer

# Other
import re
import string
import numpy as np
import pandas as pd
from sklearn.manifold import TSNE
```

```
2022-01-11 17:10:15.894812: W tensorflow/stream_executor/platform/default/dso
_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: l
ibcudart.so.11.0: cannot open shared object file: No such file or directory
2022-01-11 17:10:15.894838: I tensorflow/stream_executor/cuda/cudart_stub.cc:
29] Ignore above cudart dlerror if you do not have a GPU set up on your machi
ne.
```

Data Processing

In [5]:

```
df = pd.read_csv(jarvis.DATA_DIR + '/sentiment_analysis/yelp_labelled.txt',
                 sep = '\t',
                 names = ['text', 'stars'])
```

In [6]:

```
df.head(2)
```

Out[6]:

	text	stars
0	Wow... Loved this place.	1
1	Crust is not good.	0

In [7]:

```
df= df.dropna()
df = df[df.stars.apply(lambda x: str(x).isnumeric())]
df = df[df.stars.apply(lambda x: x !="")]
df = df[df.text.apply(lambda x: x !="")]
```

In [8]:

```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    1000 non-null   object
 1   stars   1000 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 23.4+ KB
```

Out[8]:

	stars
count	1000.00000
mean	0.50000
std	0.50025
min	0.00000
25%	0.00000
50%	0.50000
75%	1.00000
max	1.00000

In [9]:

```
df.head()
```

Out[9]:

	text	stars
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

Convert five classes into two classes (positive = 1 and negative = 0)

Since the main purpose is to identify positive or negative comments, I convert five class star category into two classes:

(1) Positive: comments with stars > 3 and

(2) Negative: comments with stars <= 3

In [10]:

```
labels = df['stars'].map(lambda x : 1 if int(x) > 3 else 0)
```

Tokenize text data

Because of the computational expenses, I use the top 20000 unique words. First, tokenize the comments then convert those into sequences. I keep 50 words to limit the number of words in each comment.

In [11]:

```
def clean_text(text):

    ## Remove puncuation
    text = text.translate(string.punctuation)

    ## Convert words to lower case and split them
    text = text.lower().split()

    ## Remove stop words
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops and len(w) >= 3]

    text = " ".join(text)

    # Clean the text
    text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r",", " ", text)
    text = re.sub(r"\.", " ", text)
    text = re.sub(r"!", " ! ", text)
    text = re.sub(r"\/", " ", text)
    text = re.sub(r"\^", " ^ ", text)
    text = re.sub(r"\+", " + ", text)
    text = re.sub(r"\-", " - ", text)
    text = re.sub(r"\=", " = ", text)
    text = re.sub(r"'", " ", text)
    text = re.sub(r"(\d+)(k)", r"\g<1>000", text)
    text = re.sub(r":", " : ", text)
    text = re.sub(r" e g ", " eg ", text)
    text = re.sub(r" b g ", " bg ", text)
    text = re.sub(r" u s ", " american ", text)
    text = re.sub(r"\0s", "0", text)
    text = re.sub(r" 9 11 ", "911", text)
    text = re.sub(r"e - mail", "email", text)
    text = re.sub(r"j k", "jk", text)
    text = re.sub(r"\s{2,}", " ", text)

    text = text.split()
    stemmer = SnowballStemmer('english')
    stemmed_words = [stemmer.stem(word) for word in text]
    text = " ".join(stemmed_words)

    return text
```

In [12]:

```
df['text'] = df['text'].map(lambda x: clean_text(x))
```

In [13]:

```
df.head(10)
```

Out[13]:

	text	stars
0	wow love place	1
1	crust good	0
2	tasti textur nasti	0
3	stop late may bank holiday rick steve recommen...	1
4	select menu great price	1
5	get angri want damn pho	0
6	honeslti tast fresh	0
7	potato like rubber could tell made ahead time ...	0
8	fri great too	1
9	great touch	1

In [14]:

```
vocabulary_size = 20000
tokenizer = Tokenizer(num_words= vocabulary_size)
tokenizer.fit_on_texts(df['text'])

sequences = tokenizer.texts_to_sequences(df['text'])
data = pad_sequences(sequences, maxlen=50)
```

In [15]:

```
print(data.shape)
```

```
(1000, 50)
```

Build neural network with LSTM

Network Architechture

The network starts with an embedding layer. The layer lets the system expand each token to a more massive vector, allowing the network to represent a word in a meaningful way. The layer takes 20000 as the first argument, which is the size of our vocabulary, and 100 as the second input parameter, which is the dimension of the embeddings. The third parameter is the input_length of 50, which is the length of each comment sequence.

In [16]:

```
model_lstm = Sequential()
model_lstm.add(Embedding(20000, 100, input_length=50))
model_lstm.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model_lstm.add(Dense(1, activation='sigmoid'))
model_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
```

```
2022-01-11 17:10:17.286554: 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 17:10:17.286596: W tensorflow/stream_executor/cuda/cuda_driver.cc:
269] failed call to cuInit: UNKNOWN ERROR (303)
2022-01-11 17:10:17.286614: 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 17:10:17.286889: 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.
```

Train the network

There are about 1.6 million comments, and it takes a while to train the model in a MacBook Pro. To save time I have used only three epochs. GPU machines can be used to accelerate the training with more time steps. I split the whole datasets as 60% for training and 40% for validation.

In [17]:

```
model_lstm.fit(data, np.array(labels), validation_split=0.4, epochs=3)
```

```
Epoch 1/3
19/19 [==============================] - 3s 62ms/step - loss: 0.3047 - accura
cy: 0.9550 - val_loss: 0.0027 - val_accuracy: 1.0000
Epoch 2/3
19/19 [==============================] - 1s 51ms/step - loss: 0.0010 - accura
cy: 1.0000 - val_loss: 3.4373e-04 - val_accuracy: 1.0000
Epoch 3/3
19/19 [==============================] - 1s 53ms/step - loss: 2.5323e-04 - ac
curacy: 1.0000 - val_loss: 1.7430e-04 - val_accuracy: 1.0000
```

Out[17]:

```
<keras.callbacks.History at 0x7f4f084fdb90>
```

Build neural network with LSTM and CNN

The LSTM model worked well. However, it takes forever to train three epochs. One way to speed up the training time is to improve the network adding “Convolutional” layer. Convolutional Neural Networks (CNN) come from image processing. They pass a “filter” over the data and calculate a higher-level representation. They have been shown to work surprisingly well for text, even though they have none of the sequence processing ability of LSTMs.

In [18]:

```
def create_conv_model():
    model_conv = Sequential()
    model_conv.add(Embedding(vocabulary_size, 100, input_length=50))
    model_conv.add(Dropout(0.2))
    model_conv.add(Conv1D(64, 5, activation='relu'))
    model_conv.add(MaxPooling1D(pool_size=4))
    model_conv.add(LSTM(100))
    model_conv.add(Dense(1, activation='sigmoid'))
    model_conv.compile(loss='binary_crossentropy', optimizer='adam', metrics=
    return model_conv
```

In [19]:

```
model_conv = create_conv_model()
model_conv.fit(data, np.array(labels), validation_split=0.4, epochs = 3)
```

```
Epoch 1/3
19/19 [==============================] - 2s 39ms/step - loss: 0.2534 - accura
cy: 0.9467 - val_loss: 4.0228e-04 - val_accuracy: 1.0000
Epoch 2/3
19/19 [==============================] - 0s 22ms/step - loss: 1.2631e-04 - ac
curacy: 1.0000 - val_loss: 4.5305e-05 - val_accuracy: 1.0000
Epoch 3/3
19/19 [==============================] - 0s 22ms/step - loss: 3.5124e-05 - ac
curacy: 1.0000 - val_loss: 3.0010e-05 - val_accuracy: 1.0000
```

Out[19]:

```
<keras.callbacks.History at 0x7f4edc613750>
```

Save processed Data

In [20]:

```
df_save = pd.DataFrame(data)
df_label = pd.DataFrame(np.array(labels))
```

In [21]:

```
result = pd.concat([df_save, df_label], axis = 1)
```

In [22]:

```
result.to_csv(jarvis.ROOT_DIR + '/train_dense_word_vectors.csv', index=False)
```

Use pre-trained Glove word embeddings

In this subsection, I want to use word embeddings from pre-trained Glove. It was trained on a dataset of one billion tokens (words) with a vocabulary of 400 thousand words. The glove has embedding vector sizes, including 50, 100, 200 and 300 dimensions. I chose the 100-dimensional version. I also want to see the model behavior in case the learned word weights do not get updated. I, therefore, set the trainable attribute for the model to be False.

Get embeddings from Glove

In [23]:

```
embeddings_index = dict()
f = open('/home/magni/ML_Root/glove_encodings/glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

```
Loaded 400000 word vectors.
```

In [24]:

```
# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocabulary_size, 100))
for word, index in tokenizer.word_index.items():
    if index > vocabulary_size - 1:
        break
    else:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector
```

Develop model

I use the same model architecture with a convolutional layer on top of the LSTM layer.

In [25]:

```
model_glove = Sequential()
model_glove.add(Embedding(vocabulary_size, 100, input_length=50, weights=[emb
model_glove.add(Dropout(0.2))
model_glove.add(Conv1D(64, 5, activation='relu'))
model_glove.add(MaxPooling1D(pool_size=4))
model_glove.add(LSTM(100))
model_glove.add(Dense(1, activation='sigmoid'))
model_glove.compile(loss='binary_crossentropy', optimizer='adam', metrics=['a
```

In [26]:

```
model_glove.fit(data, np.array(labels), validation_split=0.4, epochs = 3)
```

```
Epoch 1/3
19/19 [==============================] - 2s 33ms/step - loss: 0.2932 - accura
cy: 0.9717 - val_loss: 0.0117 - val_accuracy: 1.0000
Epoch 2/3
19/19 [==============================] - 0s 15ms/step - loss: 0.0012 - accura
cy: 1.0000 - val_loss: 1.9117e-05 - val_accuracy: 1.0000
Epoch 3/3
19/19 [==============================] - 0s 14ms/step - loss: 1.4756e-05 - ac
curacy: 1.0000 - val_loss: 1.2453e-05 - val_accuracy: 1.0000
```

Out[26]:

```
<keras.callbacks.History at 0x7f4e78b34750>
```

Word embedding visialization

In this subsection, I want to visualize word embedding weights obtained from trained models. Word embeddings with 100 dimensions are first reduced to 2 dimensions using t-SNE. Tensorflow has an excellent tool to visualize the embeddings in a great way, but here I just want to visualize the word relationship.

Get embedding weights from glove

In [27]:

```
lstm_embds = model_lstm.layers[0].get_weights()[0]
```

In [28]:

```
conv_embds = model_conv.layers[0].get_weights()[0]
```

In [29]:

```
glove_emds = model_glove.layers[0].get_weights()[0]
```

Get word list

In [30]:

```
word_list = []
for word, i in tokenizer.word_index.items():
    word_list.append(word)
```

Scatter plot of first two components of TSNE

In [31]:

```
def plot_words(data, start, stop, step):
    trace = go.Scatter(
        x = data[start:stop:step,0],
        y = data[start:stop:step, 1],
        mode = 'markers',
        text= word_list[start:stop:step]
    )
    layout = dict(title= 't-SNE 1 vs t-SNE 2',
                  yaxis = dict(title='t-SNE 2'),
                  xaxis = dict(title='t-SNE 1'),
                  hovermode= 'closest')
    fig = dict(data = [trace], layout= layout)
    py.iplot(fig)
```

1. LSTM

In [56]:

```
number_of_words = 2000
lstm_tsne_embds = TSNE(n_components=2).fit_transform(lstm_embds)
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/mani
fold/_t_sne.py:783: FutureWarning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/mani
fold/_t_sne.py:793: FutureWarning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
```

In [57]:

```
plot_words(lstm_tsne_embds, 0, number_of_words, 1)
```

2. CNN + LSTM

In [58]:

```
conv_tsne_embds = TSNE(n_components=2).fit_transform(conv_embds)
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/mani
fold/_t_sne.py:783: FutureWarning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/mani
fold/_t_sne.py:793: FutureWarning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
```

In [59]:

```
plot_words(conv_tsne_embds, 0, number_of_words, 1)
```

3. Glove

In [61]:

```
glove_tsne_embds = TSNE(n_components=2).fit_transform(glove_emds)
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/mani
fold/_t_sne.py:783: FutureWarning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/mani
fold/_t_sne.py:793: FutureWarning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
```

In [62]:

```
plot_words(glove_tsne_embds, 0, number_of_words, 1)
```

Word Embeddings from Word2Vec

In this subsection, I use word2vec to create word embeddings from the review comments. Word2vec is one algorithm for learning a word embedding from a text corpus.

In [63]:

```
from gensim.models import Word2Vec
import nltk
nltk.download('punkt')
```

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

Out[63]:

```
True
```

Tokenize the reviews coments.

In [64]:

```
df['tokenized'] = df.apply(lambda row : nltk.word_tokenize(row['text']), axis
```

In [65]:

```
df.head()
```

Out[65]:

	text	stars	tokenized
0	wow love place	1	[wow, love, place]
1	crust good	0	[crust, good]
2	tasti textur nasti	0	[tasti, textur, nasti]
3	stop late may bank holiday rick steve recommen...	1	[stop, late, may, bank, holiday, rick, steve, ...
4	select menu great price	1	[select, menu, great, price]

Train word2vec model

In [82]:

```
model_w2v = Word2Vec(df['tokenized'], vector_size=100)
```

In [89]:

```
for index, word in enumerate(model_w2v.wv.index_to_key):
    if index == 10:
        break
    print(f"word #{index}/{len(model_w2v.wv.index_to_key)} is {word}")
```

```
word #0/294 is !
word #1/294 is food
word #2/294 is place
word #3/294 is good
word #4/294 is servic
word #5/294 is great
word #6/294 is back
word #7/294 is time
word #8/294 is i
word #9/294 is like
```

In [115…

```
print (len(model_w2v.wv))
print(model_w2v)
type(model_w2v)
```

```
294
Word2Vec(vocab=294, vector_size=100, alpha=0.025)
```

Out[115…

```
gensim.models.word2vec.Word2Vec
```

In [131…

```
#Original for use in Gensim < 4.0
#X = model_w2v[model_w2v.wv.vocab]

#Attempts at understanding upgrade
#X = model_w2v.wv.index_to_key
#X = np.array(model_w2v.wv.index_to_key).reshape(1, -1)
#X = model_w2v.wv
#X = model_w2v.wv.key_to_index
X = np.array(model_w2v.wv.key_to_index)
#X = model_w2v.wv.index_to_key.keys()
#X = model_w2v.wv.get_normed_vectors()
```

In [144…

```
from sklearn.manifold import TSNE
import textscatter

XY = TSNE(model_w2v)
#figure
textscatter(XY,words)
title("Word Embedding t-SNE Plot")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_186966/4041169828.py in <module>
      1 from sklearn.manifold import TSNE
----> 2 import textscatter
      3 
      4 XY = TSNE(model_w2v)
      5 #figure

ModuleNotFoundError: No module named 'textscatter'
```

In [135…

```
#print (model_w2v.wv.get_vecattr(model_w2v.wv.index_to_key))
print (model_w2v.wv.index_to_key)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_186966/3063784243.py in <module>
----> 1 print (model_w2v.wv.get_vecattr(model_w2v.wv.index_to_key))
      2 #print (model_w2v.wv.index_to_key)

TypeError: get_vecattr() missing 1 required positional argument: 'attr'
```

Plot Word Vectors Using PCA

In [85]:

```
from sklearn.decomposition import TruncatedSVD
```

In [132…

```
tsvd = TruncatedSVD(n_components=5, n_iter=10)
result = tsvd.fit_transform(X)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_186966/1652840756.py in <module>
      1 tsvd = TruncatedSVD(n_components=5, n_iter=10)
----> 2 result = tsvd.fit_transform(X)

~/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/decomposition/
_truncated_svd.py in fit_transform(self, X, y)
    190             Reduced version of X. This will always be a dense array.
    191         """
--> 192         X = self._validate_data(X, accept_sparse=["csr", "csc"], ensu
re_min_features=2)
    193         random_state = check_random_state(self.random_state)
    194 

~/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/base.py in _va
```

```
lidate_data(self, X, y, reset, validate_separately, **check_params)
    564                 raise ValueError("Validation should be done on X, y or bo
th.")
    565             elif not no_val_X and no_val_y:
--> 566                 X = check_array(X, **check_params)
    567                 out = X
    568             elif no_val_X and not no_val_y:

~/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/utils/validati
on.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order,
copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_f
eatures, estimator)
    744                         array = array.astype(dtype, casting="unsafe", cop
y=False)
    745                     else:
--> 746                         array = np.asarray(array, order=order, dtype=dtyp
e)
    747                 except ComplexWarning as complex_warning:
    748                     raise ValueError(

~/python_env/ML1010_env2/lib64/python3.7/site-packages/numpy/core/_asarray.py
in asarray(a, dtype, order)
     81
     82     """
---> 83     return array(a, dtype, copy=False, order=order)
     84
     85
```

In [112…

```
result.shape
```

Out[112…

```
(294, 5)
```

In [114…

```
tsvd_word_list = []
words = list(model_w2v.wv)
for i, word in enumerate(words):
    tsvd_word_list.append(word)

trace = go.Scatter(
    x = result[0:number_of_words, 0],
    y = result[0:number_of_words, 1],
    mode = 'markers',
    text= tsvd_word_list[0:number_of_words]
)

layout = dict(title= 'SVD 1 vs SVD 2',
              yaxis = dict(title='SVD 2'),
              xaxis = dict(title='SVD 1'),
              hovermode= 'closest')

fig = dict(data = [trace], layout= layout)
py.iplot(fig)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/tmp/ipykernel_186966/1332749106.py in <module>
```

```
      1 tsvd_word_list = []
----> 2 words = list(model_w2v.wv)
      3 for i, word in enumerate(words):
      4     tsvd_word_list.append(word)
      5 

~/python_env/ML1010_env2/lib64/python3.7/site-packages/gensim/models/keyedvec
tors.py in __getitem__(self, key_or_keys)
    393         """
    394         if isinstance(key_or_keys, _KEY_TYPES):
--> 395             return self.get_vector(key_or_keys)
    396 
    397         return vstack([self.get_vector(key) for key in key_or_keys])

~/python_env/ML1010_env2/lib64/python3.7/site-packages/gensim/models/keyedvec
tors.py in get_vector(self, key, norm)
    436 
    437         """
--> 438         index = self.get_index(key)
    439         if norm:
    440             self.fill_norms()

~/python_env/ML1010_env2/lib64/python3.7/site-packages/gensim/models/keyedvec
tors.py in get_index(self, key, default)
    410             return default
    411         else:
--> 412             raise KeyError(f"Key '{key}' not present")
    413 
    414     def get_vector(self, key, norm=False):

KeyError: "Key '204' not present"
```

```
In [ ]:
```

In [146…

```
def tsne_plot(model):
    "Creates and TSNE model and plots it"
    labels = []
    tokens = []

    for word in model.wv.vocab:
        tokens.append(model[word])
        labels.append(word)

    tsne_model = TSNE(perplexity=40, n_components=2, init='pca', n_iter=2500,
    new_values = tsne_model.fit_transform(tokens)

    x = []
    y = []
    for value in new_values:
        x.append(value[0])
        y.append(value[1])

    plt.figure(figsize=(16, 16))
    for i in range(len(x)):
        plt.scatter(x[i],y[i])
        plt.annotate(labels[i],
                     xy=(x[i], y[i]),
                     xytext=(5, 2),
                     textcoords='offset points',
                     ha='right',
                     va='bottom')
    plt.show()

tsne_plot(model_w2v)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_186966/3499141644.py in <module>
     28     plt.show()
     29 
---> 30 tsne_plot(model_w2v)
     31 

/tmp/ipykernel_186966/3499141644.py in tsne_plot(model)
      4     tokens = []
      5 
----> 6     for word in model.wv.vocab:
      7         tokens.append(model[word])
      8         labels.append(word)

~/python_env/ML1010_env2/lib64/python3.7/site-packages/gensim/models/keyedvec
tors.py in vocab(self)
    660     def vocab(self):
    661         raise AttributeError(
--> 662             "The vocab attribute was removed from KeyedVector in Gens
im 4.0.0.\n"
    663             "Use KeyedVector's .key_to_index dict, .index_to_key lis
t, and methods "
    664             ".get_vecattr(key, attr) and .set_vecattr(key, attr, new_
val) instead.\n"
```

```
AttributeError: The vocab attribute was removed from KeyedVector in Gensim 4.
0.0.
Use KeyedVector's .key_to_index dict, .index_to_key list, and methods .get_ve
cattr(key, attr) and .set_vecattr(key, attr, new_val) instead.
See https://github.com/RaRe-Technologies/gensim/wiki/Migrating-from-Gensim-3.
x-to-4
```

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
```

Configuration

In [1]:

```
# Parameters
PROJECT_NAME = 'ML1010_Weekly'
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.
Data subdirectory 05_experiments has been created

I have set your current working directory to /home/magni/ML_Root/project_root
/ML1010_Weekly
```

```
The current time is 10:22
Hello sir. Extra caffeine may help.
```

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]:

```
#Work examples from link: https://realpython.com/python-keras-text-classifica

filepath_dict = {'yelp':   jarvis.DATA_DIR + '/sentiment_analysis/yelp_labell
                 'amazon': jarvis.DATA_DIR + '/sentiment_analysis/amazon_cell
                 'imdb':   jarvis.DATA_DIR + '/sentiment_analysis/imdb_labell

df_list = []
for source, filepath in filepath_dict.items():
    df = pd.read_csv(filepath, names=['sentence', 'label'], sep='\t')
    df['source'] = source  # Add another column filled with the source name
    df_list.append(df)

df = pd.concat(df_list)
print(df.iloc[0])
```

```
sentence    Wow... Loved this place.
label                              1
```

```
source                                yelp
Name: 0, dtype: object
```

In [6]:

```
from sklearn.model_selection import train_test_split

df_yelp = df[df['source'] == 'yelp']

sentences = df_yelp['sentence'].values
y = df_yelp['label'].values

sentences_train, sentences_test, y_train, y_test = train_test_split(
   sentences, y, test_size=0.25, random_state=1000)
```

In [9]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorizer.fit(sentences_train)

X_train = vectorizer.transform(sentences_train)
X_test  = vectorizer.transform(sentences_test)
X_train
```

Out[9]:

```
<750x1714 sparse matrix of type '<class 'numpy.int64'>'
        with 7368 stored elements in Compressed Sparse Row format>
```

In [10]:

```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression()
classifier.fit(X_train, y_train)
score = classifier.score(X_test, y_test)

print("Accuracy:", score)
```

```
Accuracy: 0.796
```

In [11]:

```
for source in df['source'].unique():
    df_source = df[df['source'] == source]
    sentences = df_source['sentence'].values
    y = df_source['label'].values

    sentences_train, sentences_test, y_train, y_test = train_test_split(
        sentences, y, test_size=0.25, random_state=1000)

    vectorizer = CountVectorizer()
    vectorizer.fit(sentences_train)
    X_train = vectorizer.transform(sentences_train)
    X_test  = vectorizer.transform(sentences_test)

    classifier = LogisticRegression()
    classifier.fit(X_train, y_train)
    score = classifier.score(X_test, y_test)
    print('Accuracy for {} data: {:.4f}'.format(source, score))
```

```
Accuracy for yelp data: 0.7960
Accuracy for amazon data: 0.7960
Accuracy for imdb data: 0.7487
```

In [12]:

```
from keras.models import Sequential
from keras import layers

input_dim = X_train.shape[1]  # Number of features

model = Sequential()
model.add(layers.Dense(10, input_dim=input_dim, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
2022-01-11 10:40:57.916349: W tensorflow/stream_executor/platform/default/dso
_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: l
ibcudart.so.11.0: cannot open shared object file: No such file or directory
2022-01-11 10:40:57.916378: I tensorflow/stream_executor/cuda/cudart_stub.cc:
29] Ignore above cudart dlerror if you do not have a GPU set up on your machi
ne.
2022-01-11 10:40:58.829406: 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 10:40:58.829438: W tensorflow/stream_executor/cuda/cuda_driver.cc:
269] failed call to cuInit: UNKNOWN ERROR (303)
2022-01-11 10:40:58.829452: 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 10:40:58.829653: 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.
```

In [13]:

```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 10)                25060     
                                                                 
 dense_1 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 25,071
Trainable params: 25,071
Non-trainable params: 0
_________________________________________________________________
```

In [28]:

```
history = model.fit(X_train, y_train,
                    epochs=100,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
```

```
Epoch 1/100
57/57 [==============================] - 0s 2ms/step - loss: 1.0055e-05 - acc
uracy: 1.0000 - val_loss: 1.5872 - val_accuracy: 0.7861
Epoch 2/100
57/57 [==============================] - 0s 2ms/step - loss: 9.7718e-06 - acc
uracy: 1.0000 - val_loss: 1.5870 - val_accuracy: 0.7861
Epoch 3/100
57/57 [==============================] - 0s 2ms/step - loss: 9.4272e-06 - acc
uracy: 1.0000 - val_loss: 1.5970 - val_accuracy: 0.7861
Epoch 4/100
57/57 [==============================] - 0s 2ms/step - loss: 9.1060e-06 - acc
uracy: 1.0000 - val_loss: 1.6004 - val_accuracy: 0.7861
Epoch 5/100
57/57 [==============================] - 0s 2ms/step - loss: 8.8834e-06 - acc
uracy: 1.0000 - val_loss: 1.6057 - val_accuracy: 0.7861
Epoch 6/100
57/57 [==============================] - 0s 2ms/step - loss: 8.5844e-06 - acc
uracy: 1.0000 - val_loss: 1.6052 - val_accuracy: 0.7861
Epoch 7/100
57/57 [==============================] - 0s 2ms/step - loss: 8.2953e-06 - acc
uracy: 1.0000 - val_loss: 1.6090 - val_accuracy: 0.7861
Epoch 8/100
57/57 [==============================] - 0s 2ms/step - loss: 8.0686e-06 - acc
uracy: 1.0000 - val_loss: 1.6170 - val_accuracy: 0.7861
Epoch 9/100
57/57 [==============================] - 0s 2ms/step - loss: 7.8163e-06 - acc
uracy: 1.0000 - val_loss: 1.6180 - val_accuracy: 0.7861
Epoch 10/100
57/57 [==============================] - 0s 2ms/step - loss: 7.5623e-06 - acc
uracy: 1.0000 - val_loss: 1.6205 - val_accuracy: 0.7861
Epoch 11/100
57/57 [==============================] - 0s 2ms/step - loss: 7.3251e-06 - acc
```

```
uracy: 1.0000 - val_loss: 1.6215 - val_accuracy: 0.7861
Epoch 12/100
57/57 [==============================] - 0s 2ms/step - loss: 7.1410e-06 - acc
uracy: 1.0000 - val_loss: 1.6267 - val_accuracy: 0.7861
Epoch 13/100
57/57 [==============================] - 0s 2ms/step - loss: 6.9032e-06 - acc
uracy: 1.0000 - val_loss: 1.6323 - val_accuracy: 0.7861
Epoch 14/100
57/57 [==============================] - 0s 2ms/step - loss: 6.6716e-06 - acc
uracy: 1.0000 - val_loss: 1.6369 - val_accuracy: 0.7861
Epoch 15/100
57/57 [==============================] - 0s 2ms/step - loss: 6.4915e-06 - acc
uracy: 1.0000 - val_loss: 1.6462 - val_accuracy: 0.7861
Epoch 16/100
57/57 [==============================] - 0s 2ms/step - loss: 6.2846e-06 - acc
uracy: 1.0000 - val_loss: 1.6456 - val_accuracy: 0.7861
Epoch 17/100
57/57 [==============================] - 0s 2ms/step - loss: 6.0810e-06 - acc
uracy: 1.0000 - val_loss: 1.6481 - val_accuracy: 0.7861
Epoch 18/100
57/57 [==============================] - 0s 2ms/step - loss: 5.9069e-06 - acc
uracy: 1.0000 - val_loss: 1.6537 - val_accuracy: 0.7861
Epoch 19/100
57/57 [==============================] - 0s 2ms/step - loss: 5.7308e-06 - acc
uracy: 1.0000 - val_loss: 1.6582 - val_accuracy: 0.7861
Epoch 20/100
57/57 [==============================] - 0s 2ms/step - loss: 5.5437e-06 - acc
uracy: 1.0000 - val_loss: 1.6636 - val_accuracy: 0.7861
Epoch 21/100
57/57 [==============================] - 0s 2ms/step - loss: 5.3650e-06 - acc
uracy: 1.0000 - val_loss: 1.6661 - val_accuracy: 0.7861
Epoch 22/100
57/57 [==============================] - 0s 2ms/step - loss: 5.2201e-06 - acc
uracy: 1.0000 - val_loss: 1.6669 - val_accuracy: 0.7861
Epoch 23/100
57/57 [==============================] - 0s 2ms/step - loss: 5.0480e-06 - acc
uracy: 1.0000 - val_loss: 1.6714 - val_accuracy: 0.7861
Epoch 24/100
57/57 [==============================] - 0s 2ms/step - loss: 4.8799e-06 - acc
uracy: 1.0000 - val_loss: 1.6806 - val_accuracy: 0.7861
Epoch 25/100
57/57 [==============================] - 0s 2ms/step - loss: 4.7613e-06 - acc
uracy: 1.0000 - val_loss: 1.6818 - val_accuracy: 0.7861
Epoch 26/100
57/57 [==============================] - 0s 2ms/step - loss: 4.5957e-06 - acc
uracy: 1.0000 - val_loss: 1.6855 - val_accuracy: 0.7861
Epoch 27/100
57/57 [==============================] - 0s 2ms/step - loss: 4.4456e-06 - acc
uracy: 1.0000 - val_loss: 1.6872 - val_accuracy: 0.7861
Epoch 28/100
57/57 [==============================] - 0s 2ms/step - loss: 4.3308e-06 - acc
uracy: 1.0000 - val_loss: 1.6954 - val_accuracy: 0.7861
Epoch 29/100
57/57 [==============================] - 0s 2ms/step - loss: 4.1918e-06 - acc
uracy: 1.0000 - val_loss: 1.6979 - val_accuracy: 0.7861
Epoch 30/100
57/57 [==============================] - 0s 2ms/step - loss: 4.0555e-06 - acc
uracy: 1.0000 - val_loss: 1.6995 - val_accuracy: 0.7861
Epoch 31/100
```

```
57/57 [==============================] - 0s 2ms/step - loss: 3.9290e-06 - acc
uracy: 1.0000 - val_loss: 1.7066 - val_accuracy: 0.7861
Epoch 32/100
57/57 [==============================] - 0s 2ms/step - loss: 3.8240e-06 - acc
uracy: 1.0000 - val_loss: 1.7101 - val_accuracy: 0.7861
Epoch 33/100
57/57 [==============================] - 0s 2ms/step - loss: 3.7072e-06 - acc
uracy: 1.0000 - val_loss: 1.7114 - val_accuracy: 0.7861
Epoch 34/100
57/57 [==============================] - 0s 2ms/step - loss: 3.5806e-06 - acc
uracy: 1.0000 - val_loss: 1.7163 - val_accuracy: 0.7861
Epoch 35/100
57/57 [==============================] - 0s 2ms/step - loss: 3.4895e-06 - acc
uracy: 1.0000 - val_loss: 1.7225 - val_accuracy: 0.7861
Epoch 36/100
57/57 [==============================] - 0s 2ms/step - loss: 3.3782e-06 - acc
uracy: 1.0000 - val_loss: 1.7261 - val_accuracy: 0.7861
Epoch 37/100
57/57 [==============================] - 0s 2ms/step - loss: 3.2666e-06 - acc
uracy: 1.0000 - val_loss: 1.7307 - val_accuracy: 0.7861
Epoch 38/100
57/57 [==============================] - 0s 2ms/step - loss: 3.1744e-06 - acc
uracy: 1.0000 - val_loss: 1.7351 - val_accuracy: 0.7861
Epoch 39/100
57/57 [==============================] - 0s 2ms/step - loss: 3.0813e-06 - acc
uracy: 1.0000 - val_loss: 1.7409 - val_accuracy: 0.7861
Epoch 40/100
57/57 [==============================] - 0s 2ms/step - loss: 2.9803e-06 - acc
uracy: 1.0000 - val_loss: 1.7427 - val_accuracy: 0.7861
Epoch 41/100
57/57 [==============================] - 0s 2ms/step - loss: 2.8907e-06 - acc
uracy: 1.0000 - val_loss: 1.7501 - val_accuracy: 0.7861
Epoch 42/100
57/57 [==============================] - 0s 2ms/step - loss: 2.8153e-06 - acc
uracy: 1.0000 - val_loss: 1.7498 - val_accuracy: 0.7861
Epoch 43/100
57/57 [==============================] - 0s 2ms/step - loss: 2.7227e-06 - acc
uracy: 1.0000 - val_loss: 1.7555 - val_accuracy: 0.7861
Epoch 44/100
57/57 [==============================] - 0s 2ms/step - loss: 2.6407e-06 - acc
uracy: 1.0000 - val_loss: 1.7603 - val_accuracy: 0.7861
Epoch 45/100
57/57 [==============================] - 0s 2ms/step - loss: 2.5620e-06 - acc
uracy: 1.0000 - val_loss: 1.7655 - val_accuracy: 0.7861
Epoch 46/100
57/57 [==============================] - 0s 2ms/step - loss: 2.4839e-06 - acc
uracy: 1.0000 - val_loss: 1.7663 - val_accuracy: 0.7861
Epoch 47/100
57/57 [==============================] - 0s 2ms/step - loss: 2.4040e-06 - acc
uracy: 1.0000 - val_loss: 1.7704 - val_accuracy: 0.7861
Epoch 48/100
57/57 [==============================] - 0s 2ms/step - loss: 2.3377e-06 - acc
uracy: 1.0000 - val_loss: 1.7780 - val_accuracy: 0.7861
Epoch 49/100
57/57 [==============================] - 0s 2ms/step - loss: 2.2698e-06 - acc
uracy: 1.0000 - val_loss: 1.7790 - val_accuracy: 0.7861
Epoch 50/100
57/57 [==============================] - 0s 2ms/step - loss: 2.1987e-06 - acc
uracy: 1.0000 - val_loss: 1.7801 - val_accuracy: 0.7861
```

```
Epoch 51/100
57/57 [==============================] - 0s 2ms/step - loss: 2.1318e-06 - acc
uracy: 1.0000 - val_loss: 1.7878 - val_accuracy: 0.7861
Epoch 52/100
57/57 [==============================] - 0s 2ms/step - loss: 2.0688e-06 - acc
uracy: 1.0000 - val_loss: 1.7883 - val_accuracy: 0.7861
Epoch 53/100
57/57 [==============================] - 0s 2ms/step - loss: 2.0057e-06 - acc
uracy: 1.0000 - val_loss: 1.7950 - val_accuracy: 0.7861
Epoch 54/100
57/57 [==============================] - 0s 2ms/step - loss: 1.9447e-06 - acc
uracy: 1.0000 - val_loss: 1.7965 - val_accuracy: 0.7861
Epoch 55/100
57/57 [==============================] - 0s 2ms/step - loss: 1.8876e-06 - acc
uracy: 1.0000 - val_loss: 1.7956 - val_accuracy: 0.7807
Epoch 56/100
57/57 [==============================] - ETA: 0s - loss: 2.1244e-06 - accurac
y: 1.00 - 0s 2ms/step - loss: 1.8343e-06 - accuracy: 1.0000 - val_loss: 1.802
3 - val_accuracy: 0.7807
Epoch 57/100
57/57 [==============================] - 0s 2ms/step - loss: 1.7693e-06 - acc
uracy: 1.0000 - val_loss: 1.8083 - val_accuracy: 0.7807
Epoch 58/100
57/57 [==============================] - 0s 2ms/step - loss: 1.7282e-06 - acc
uracy: 1.0000 - val_loss: 1.8179 - val_accuracy: 0.7861
Epoch 59/100
57/57 [==============================] - 0s 2ms/step - loss: 1.6712e-06 - acc
uracy: 1.0000 - val_loss: 1.8219 - val_accuracy: 0.7807
Epoch 60/100
57/57 [==============================] - 0s 2ms/step - loss: 1.6193e-06 - acc
uracy: 1.0000 - val_loss: 1.8269 - val_accuracy: 0.7807
Epoch 61/100
57/57 [==============================] - 0s 2ms/step - loss: 1.5679e-06 - acc
uracy: 1.0000 - val_loss: 1.8319 - val_accuracy: 0.7807
Epoch 62/100
57/57 [==============================] - 0s 2ms/step - loss: 1.5245e-06 - acc
uracy: 1.0000 - val_loss: 1.8323 - val_accuracy: 0.7807
Epoch 63/100
57/57 [==============================] - 0s 2ms/step - loss: 1.4795e-06 - acc
uracy: 1.0000 - val_loss: 1.8371 - val_accuracy: 0.7807
Epoch 64/100
57/57 [==============================] - 0s 2ms/step - loss: 1.4292e-06 - acc
uracy: 1.0000 - val_loss: 1.8456 - val_accuracy: 0.7807
Epoch 65/100
57/57 [==============================] - 0s 2ms/step - loss: 1.3941e-06 - acc
uracy: 1.0000 - val_loss: 1.8469 - val_accuracy: 0.7807
Epoch 66/100
57/57 [==============================] - 0s 2ms/step - loss: 1.3481e-06 - acc
uracy: 1.0000 - val_loss: 1.8532 - val_accuracy: 0.7807
Epoch 67/100
57/57 [==============================] - 0s 2ms/step - loss: 1.3043e-06 - acc
uracy: 1.0000 - val_loss: 1.8542 - val_accuracy: 0.7807
Epoch 68/100
57/57 [==============================] - 0s 2ms/step - loss: 1.2675e-06 - acc
uracy: 1.0000 - val_loss: 1.8560 - val_accuracy: 0.7807
Epoch 69/100
57/57 [==============================] - 0s 2ms/step - loss: 1.2301e-06 - acc
uracy: 1.0000 - val_loss: 1.8674 - val_accuracy: 0.7807
Epoch 70/100
```

```
57/57 [==============================] - 0s 2ms/step - loss: 1.1909e-06 - acc
uracy: 1.0000 - val_loss: 1.8653 - val_accuracy: 0.7807
Epoch 71/100
57/57 [==============================] - 0s 2ms/step - loss: 1.1558e-06 - acc
uracy: 1.0000 - val_loss: 1.8727 - val_accuracy: 0.7807
Epoch 72/100
57/57 [==============================] - 0s 2ms/step - loss: 1.1213e-06 - acc
uracy: 1.0000 - val_loss: 1.8784 - val_accuracy: 0.7807
Epoch 73/100
57/57 [==============================] - 0s 2ms/step - loss: 1.0809e-06 - acc
uracy: 1.0000 - val_loss: 1.8924 - val_accuracy: 0.7807
Epoch 74/100
57/57 [==============================] - 0s 2ms/step - loss: 1.0456e-06 - acc
uracy: 1.0000 - val_loss: 1.8897 - val_accuracy: 0.7754
Epoch 75/100
57/57 [==============================] - 0s 2ms/step - loss: 1.0168e-06 - acc
uracy: 1.0000 - val_loss: 1.8905 - val_accuracy: 0.7754
Epoch 76/100
57/57 [==============================] - 0s 2ms/step - loss: 9.8682e-07 - acc
uracy: 1.0000 - val_loss: 1.9009 - val_accuracy: 0.7807
Epoch 77/100
57/57 [==============================] - 0s 2ms/step - loss: 9.5769e-07 - acc
uracy: 1.0000 - val_loss: 1.9067 - val_accuracy: 0.7807
Epoch 78/100
57/57 [==============================] - 0s 2ms/step - loss: 9.2829e-07 - acc
uracy: 1.0000 - val_loss: 1.9072 - val_accuracy: 0.7807
Epoch 79/100
57/57 [==============================] - 0s 2ms/step - loss: 8.9991e-07 - acc
uracy: 1.0000 - val_loss: 1.9131 - val_accuracy: 0.7807
Epoch 80/100
57/57 [==============================] - 0s 2ms/step - loss: 8.7384e-07 - acc
uracy: 1.0000 - val_loss: 1.9179 - val_accuracy: 0.7807
Epoch 81/100
57/57 [==============================] - 0s 2ms/step - loss: 8.4785e-07 - acc
uracy: 1.0000 - val_loss: 1.9206 - val_accuracy: 0.7807
Epoch 82/100
57/57 [==============================] - 0s 2ms/step - loss: 8.2445e-07 - acc
uracy: 1.0000 - val_loss: 1.9244 - val_accuracy: 0.7807
Epoch 83/100
57/57 [==============================] - 0s 2ms/step - loss: 7.9926e-07 - acc
uracy: 1.0000 - val_loss: 1.9257 - val_accuracy: 0.7807
Epoch 84/100
57/57 [==============================] - 0s 2ms/step - loss: 7.7681e-07 - acc
uracy: 1.0000 - val_loss: 1.9322 - val_accuracy: 0.7807
Epoch 85/100
57/57 [==============================] - 0s 2ms/step - loss: 7.5231e-07 - acc
uracy: 1.0000 - val_loss: 1.9386 - val_accuracy: 0.7807
Epoch 86/100
57/57 [==============================] - 0s 2ms/step - loss: 7.3058e-07 - acc
uracy: 1.0000 - val_loss: 1.9401 - val_accuracy: 0.7807
Epoch 87/100
57/57 [==============================] - 0s 2ms/step - loss: 7.0777e-07 - acc
uracy: 1.0000 - val_loss: 1.9431 - val_accuracy: 0.7807
Epoch 88/100
57/57 [==============================] - 0s 2ms/step - loss: 6.8771e-07 - acc
uracy: 1.0000 - val_loss: 1.9455 - val_accuracy: 0.7807
Epoch 89/100
57/57 [==============================] - 0s 2ms/step - loss: 6.6821e-07 - acc
uracy: 1.0000 - val_loss: 1.9508 - val_accuracy: 0.7807
```

```
Epoch 90/100
57/57 [==============================] - 0s 2ms/step - loss: 6.4786e-07 - acc
uracy: 1.0000 - val_loss: 1.9500 - val_accuracy: 0.7807
Epoch 91/100
57/57 [==============================] - 0s 2ms/step - loss: 6.2936e-07 - acc
uracy: 1.0000 - val_loss: 1.9569 - val_accuracy: 0.7807
Epoch 92/100
57/57 [==============================] - 0s 1ms/step - loss: 6.1042e-07 - acc
uracy: 1.0000 - val_loss: 1.9574 - val_accuracy: 0.7807
Epoch 93/100
57/57 [==============================] - 0s 2ms/step - loss: 5.9473e-07 - acc
uracy: 1.0000 - val_loss: 1.9608 - val_accuracy: 0.7807
Epoch 94/100
57/57 [==============================] - 0s 2ms/step - loss: 5.7610e-07 - acc
uracy: 1.0000 - val_loss: 1.9628 - val_accuracy: 0.7807
Epoch 95/100
57/57 [==============================] - 0s 1ms/step - loss: 5.5891e-07 - acc
uracy: 1.0000 - val_loss: 1.9701 - val_accuracy: 0.7807
Epoch 96/100
57/57 [==============================] - 0s 2ms/step - loss: 5.4412e-07 - acc
uracy: 1.0000 - val_loss: 1.9797 - val_accuracy: 0.7807
Epoch 97/100
57/57 [==============================] - 0s 2ms/step - loss: 5.2754e-07 - acc
uracy: 1.0000 - val_loss: 1.9752 - val_accuracy: 0.7807
Epoch 98/100
57/57 [==============================] - 0s 2ms/step - loss: 5.1132e-07 - acc
uracy: 1.0000 - val_loss: 1.9790 - val_accuracy: 0.7807
Epoch 99/100
57/57 [==============================] - 0s 2ms/step - loss: 5.0065e-07 - acc
uracy: 1.0000 - val_loss: 2.0096 - val_accuracy: 0.7754
Epoch 100/100
57/57 [==============================] - 0s 2ms/step - loss: 4.8158e-07 - acc
uracy: 1.0000 - val_loss: 2.0228 - val_accuracy: 0.7754
```

In [40]:

```
#Clear session before retraining or you will start with the computed weights
from keras.backend import clear_session
clear_session()
```

In [29]:

```
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

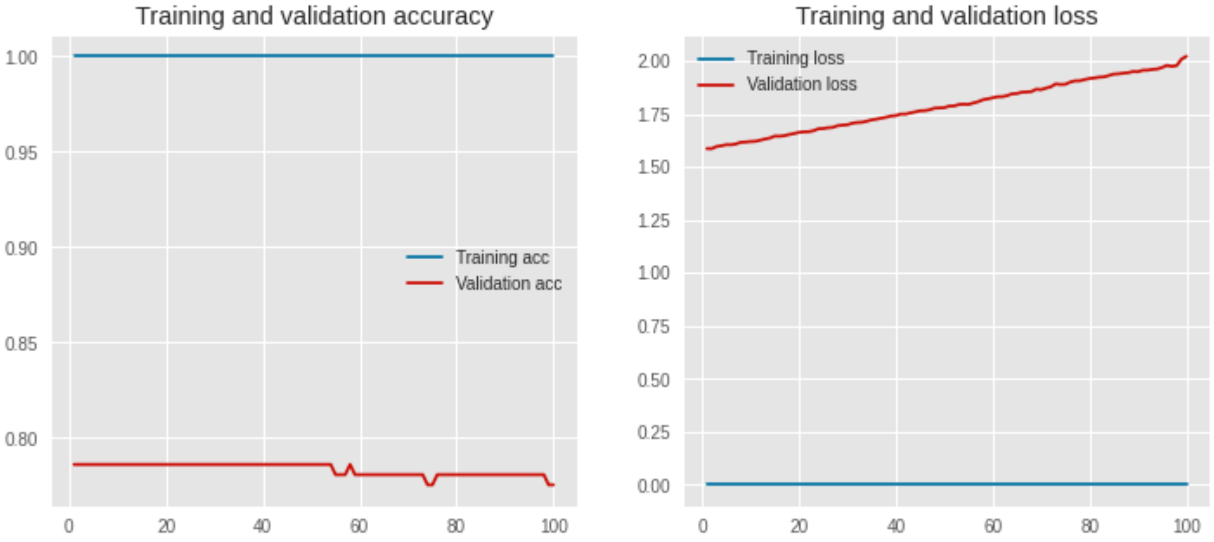
```
Training Accuracy: 1.0000
Testing Accuracy:  0.7754
```

In [30]:

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, 'b', label='Training acc')
    plt.plot(x, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, loss, 'b', label='Training loss')
    plt.plot(x, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
```

In [31]:

```
plot_history(history)
```



In [32]:

```
from keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(sentences_train)

X_train = tokenizer.texts_to_sequences(sentences_train)
X_test = tokenizer.texts_to_sequences(sentences_test)

vocab_size = len(tokenizer.word_index) + 1  # Adding 1 because of reserved 0

print(sentences_train[2])
print(X_train[2])
```

```
I am a fan of his ... This movie sucked really bad.
[7, 150, 2, 932, 4, 49, 6, 11, 563, 45, 30]
```

In [35]:

```python
for word in ['the', 'all', 'fan', 'sucked']:
    print('{}: {}'.format(word, tokenizer.word_index[word]))
```

```
the: 1
all: 27
fan: 932
sucked: 563
```

In [37]:

```python
from keras.preprocessing.sequence import pad_sequences

maxlen = 100

X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

print(X_train[1, :])
```

```
[  7 310  97   8 117   3 117   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

In [43]:

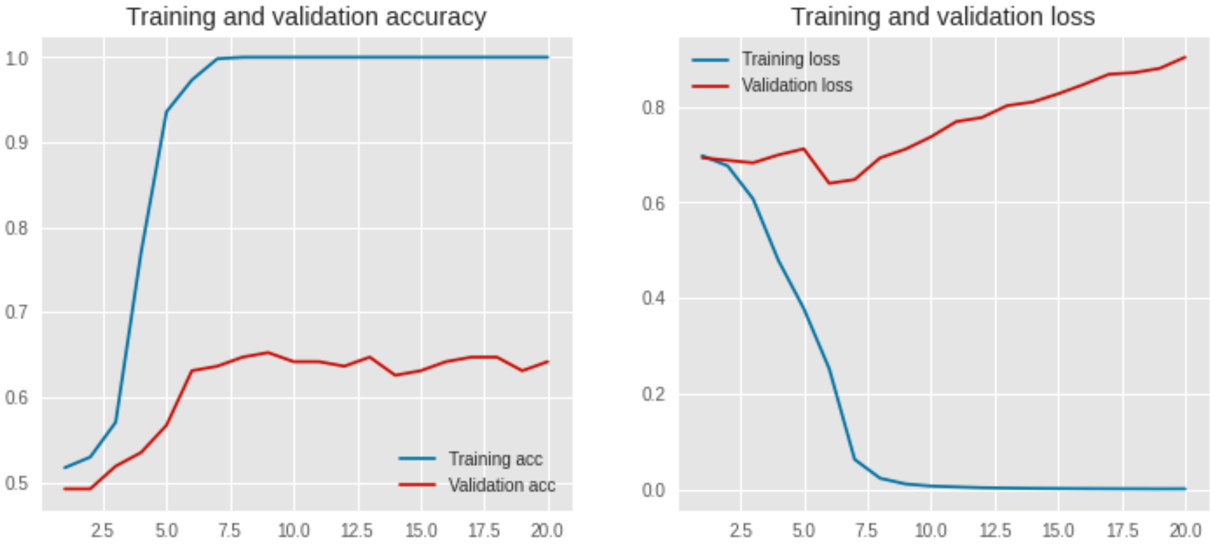
```python
from keras.models import Sequential
from keras import layers

embedding_dim = 50

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=maxlen))
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 50)           128750

 flatten (Flatten)           (None, 5000)              0

 dense (Dense)               (None, 10)                50010

 dense_1 (Dense)             (None, 1)                 11

=================================================================
Total params: 178,771
Trainable params: 178,771
Non-trainable params: 0
```

In [44]:

```
history = model.fit(X_train, y_train,
                    epochs=20,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

```
Training Accuracy: 1.0000
Testing Accuracy:  0.6417
```



In [48]:

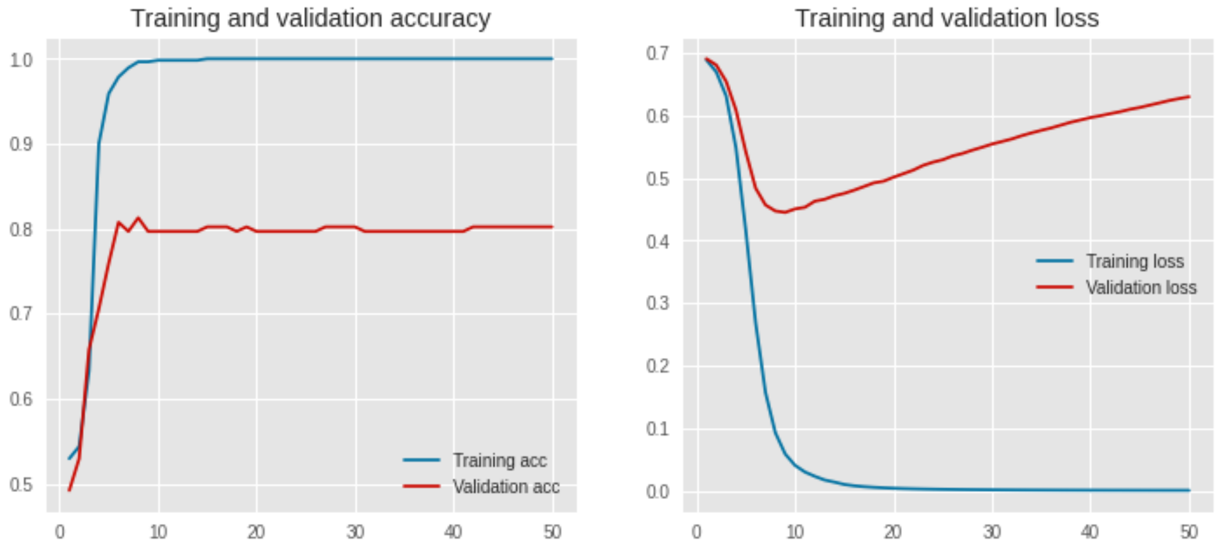
```
from keras.models import Sequential
from keras import layers

embedding_dim = 50

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=maxlen))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 100, 50)           128750

 global_max_pooling1d_1 (Glo  (None, 50)               0
 balMaxPooling1D)

 dense_4 (Dense)             (None, 10)                510

 dense_5 (Dense)             (None, 1)                 11

=================================================================
Total params: 129,271
Trainable params: 129,271
Non-trainable params: 0
```

In [49]:

```
history = model.fit(X_train, y_train,
                    epochs=50,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

```
Training Accuracy: 1.0000
Testing Accuracy:  0.8021
```



In [50]:

```
import numpy as np

def create_embedding_matrix(filepath, word_index, embedding_dim):
    vocab_size = len(word_index) + 1  # Adding again 1 because of reserved 0
    embedding_matrix = np.zeros((vocab_size, embedding_dim))

    with open(filepath) as f:
        for line in f:
            word, *vector = line.split()
            if word in word_index:
                idx = word_index[word]
                embedding_matrix[idx] = np.array(
                    vector, dtype=np.float32)[:embedding_dim]

    return embedding_matrix
```

In [51]:

```
embedding_dim = 50
embedding_matrix = create_embedding_matrix(
    '/home/magni/ML_Root/glove_encodings/glove.6B.50d.txt',
    tokenizer.word_index, embedding_dim)
```

In [52]:

```
nonzero_elements = np.count_nonzero(np.count_nonzero(embedding_matrix, axis=1
nonzero_elements / vocab_size
```

Out[52]:

```
0.9522330097087378
```

In [53]:

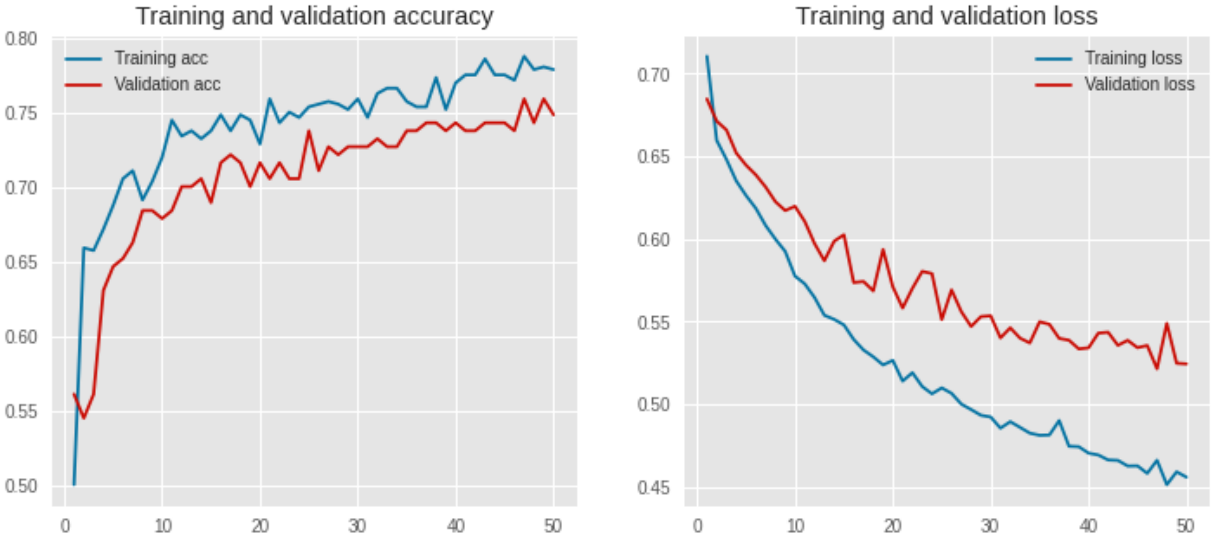
```
model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim,
                           weights=[embedding_matrix],
                           input_length=maxlen,
                           trainable=False))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 100, 50)           128750

 global_max_pooling1d_2 (Glo  (None, 50)               0
 balMaxPooling1D)

 dense_6 (Dense)             (None, 10)                510

 dense_7 (Dense)             (None, 1)                 11

=================================================================
Total params: 129,271
Trainable params: 521
Non-trainable params: 128,750
```

In [54]:

```python
history = model.fit(X_train, y_train,
                    epochs=50,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

```
Training Accuracy: 0.7879
Testing Accuracy:  0.7487
```



In [55]:

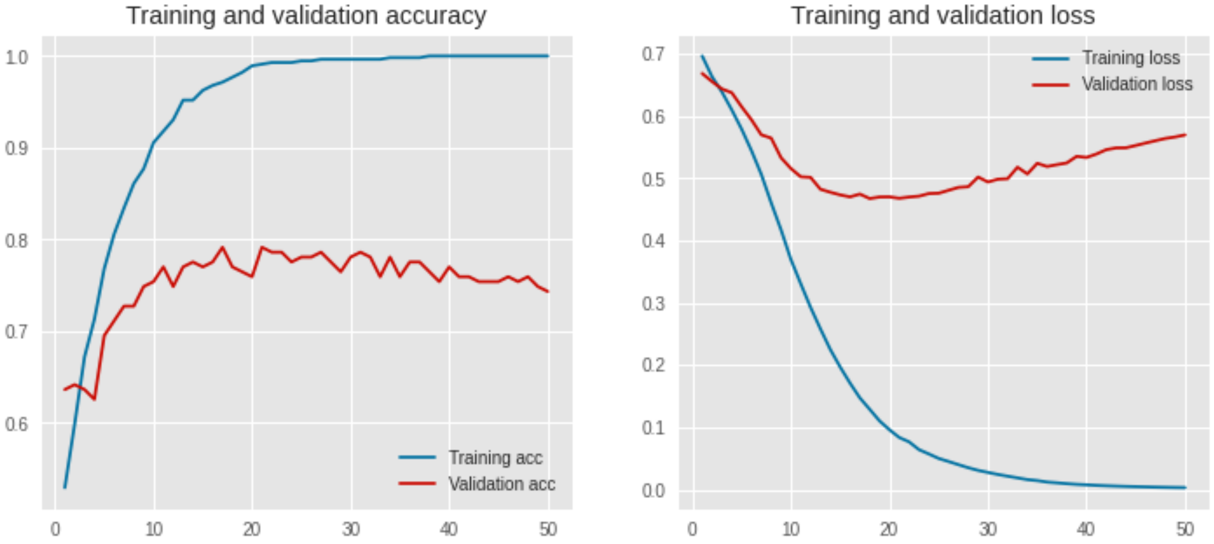
```python
model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim,
                           weights=[embedding_matrix],
                           input_length=maxlen,
                           trainable=True))
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_4"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 100, 50)           128750

 global_max_pooling1d_3 (Glo  (None, 50)               0
 balMaxPooling1D)
 dense_8 (Dense)             (None, 10)                510

 dense_9 (Dense)             (None, 1)                 11

=================================================================
Total params: 129,271
Trainable params: 129,271
Non-trainable params: 0
```

In [56]:

```
history = model.fit(X_train, y_train,
                    epochs=50,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

```
Training Accuracy: 1.0000
Testing Accuracy:  0.7433
```



In [57]:

```
#Now do the CNN
embedding_dim = 100

model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim, input_length=maxlen))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

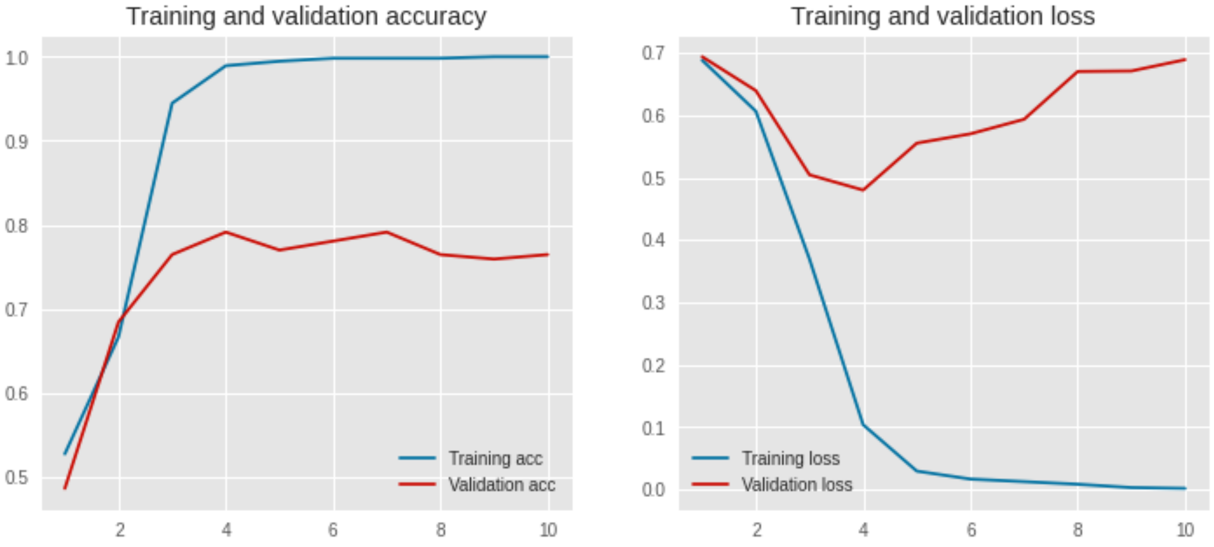
```
Model: "sequential_5"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_5 (Embedding)     (None, 100, 100)          257500    
                                                                 
 conv1d (Conv1D)             (None, 96, 128)           64128     
                                                                 
 global_max_pooling1d_4 (Glo  (None, 128)              0         
 balMaxPooling1D)                                                
                                                                 
 dense_10 (Dense)            (None, 10)                1290      
                                                                 
 dense_11 (Dense)            (None, 1)                 11        
                                                                 
=================================================================
Total params: 322,929
Trainable params: 322,929
Non-trainable params: 0
```

In [58]:

```
history = model.fit(X_train, y_train,
                    epochs=10,
                    verbose=False,
                    validation_data=(X_test, y_test),
                    batch_size=10)
loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

```
Training Accuracy: 1.0000
Testing Accuracy:  0.7647
```



In [59]:

```
#GRID SEARCH!!!
#Keras classifier and grid search
def create_model(num_filters, kernel_size, vocab_size, embedding_dim, maxlen)
    model = Sequential()
    model.add(layers.Embedding(vocab_size, embedding_dim, input_length=maxlen
    model.add(layers.Conv1D(num_filters, kernel_size, activation='relu'))
    model.add(layers.GlobalMaxPooling1D())
    model.add(layers.Dense(10, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model
```

In [61]:

```
param_grid = dict(num_filters=[32, 64, 128],
                  kernel_size=[3, 5, 7],
                  vocab_size=[5000],
                  embedding_dim=[50],
                  maxlen=[100])
```

In [63]:

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import RandomizedSearchCV

# Main settings
epochs = 20
embedding_dim = 50
maxlen = 100
output_file = jarvis.DATA_DIR + '/wk5_reading2.output.txt'

# Run grid search for each source (yelp, amazon, imdb)
for source, frame in df.groupby('source'):
    print('Running grid search for data set :', source)
    sentences = df['sentence'].values
    y = df['label'].values

    # Train-test split
    sentences_train, sentences_test, y_train, y_test = train_test_split(
        sentences, y, test_size=0.25, random_state=1000)

    # Tokenize words
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(sentences_train)
    X_train = tokenizer.texts_to_sequences(sentences_train)
    X_test = tokenizer.texts_to_sequences(sentences_test)

    # Adding 1 because of reserved 0 index
    vocab_size = len(tokenizer.word_index) + 1

    # Pad sequences with zeros
    X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
    X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

    # Parameter grid for grid search
    param_grid = dict(num_filters=[32, 64, 128],
                      kernel_size=[3, 5, 7],
                      vocab_size=[vocab_size],
                      embedding_dim=[embedding_dim],
                      maxlen=[maxlen])
    model = KerasClassifier(build_fn=create_model,
                            epochs=epochs, batch_size=10,
                            verbose=False)
    grid = RandomizedSearchCV(estimator=model, param_distributions=param_grid
                              cv=4, verbose=1, n_iter=5)
    grid_result = grid.fit(X_train, y_train)

    # Evaluate testing set
    test_accuracy = grid.score(X_test, y_test)

    # Save and evaluate results
    prompt = input(f'finished {source}; write to file and proceed? [y/n]')
    if prompt.lower() not in {'y', 'true', 'yes'}:
        break
    with open(output_file, 'a') as f:
        s = ('Running {} data set\nBest Accuracy : '
             '{:.4f}\n{}\nTest Accuracy : {:.4f}\n\n')
        output_string = s.format(
            source,
            grid_result.best_score_,
Running grid search for data set : amazon
Fitting 4 folds for each of 5 candidates, totalling 20 fits
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/ipykernel_la
uncher.py:41: DeprecationWarning: KerasClassifier is deprecated, use Sci-Kera
s (https://github.com/adriangb/scikeras) instead.
```

```
Running amazon data set
Best Accuracy : 0.8229
{'vocab_size': 4603, 'num_filters': 32, 'maxlen': 100, 'kernel_size': 3, 'emb
edding_dim': 50}
Test Accuracy : 0.8326


Running grid search for data set : imdb
Fitting 4 folds for each of 5 candidates, totalling 20 fits
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/ipykernel_la
uncher.py:41: DeprecationWarning: KerasClassifier is deprecated, use Sci-Kera
s (https://github.com/adriangb/scikeras) instead.
```

```
Running imdb data set
Best Accuracy : 0.8151
{'vocab_size': 4603, 'num_filters': 64, 'maxlen': 100, 'kernel_size': 5, 'emb
edding_dim': 50}
Test Accuracy : 0.8326


Running grid search for data set : yelp
Fitting 4 folds for each of 5 candidates, totalling 20 fits
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/ipykernel_la
uncher.py:41: DeprecationWarning: KerasClassifier is deprecated, use Sci-Kera
s (https://github.com/adriangb/scikeras) instead.
```

```
Running yelp data set
Best Accuracy : 0.8195
{'vocab_size': 4603, 'num_filters': 64, 'maxlen': 100, 'kernel_size': 3, 'emb
edding_dim': 50}
Test Accuracy : 0.8282
```