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]:

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

In [29]:

```python
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))
```

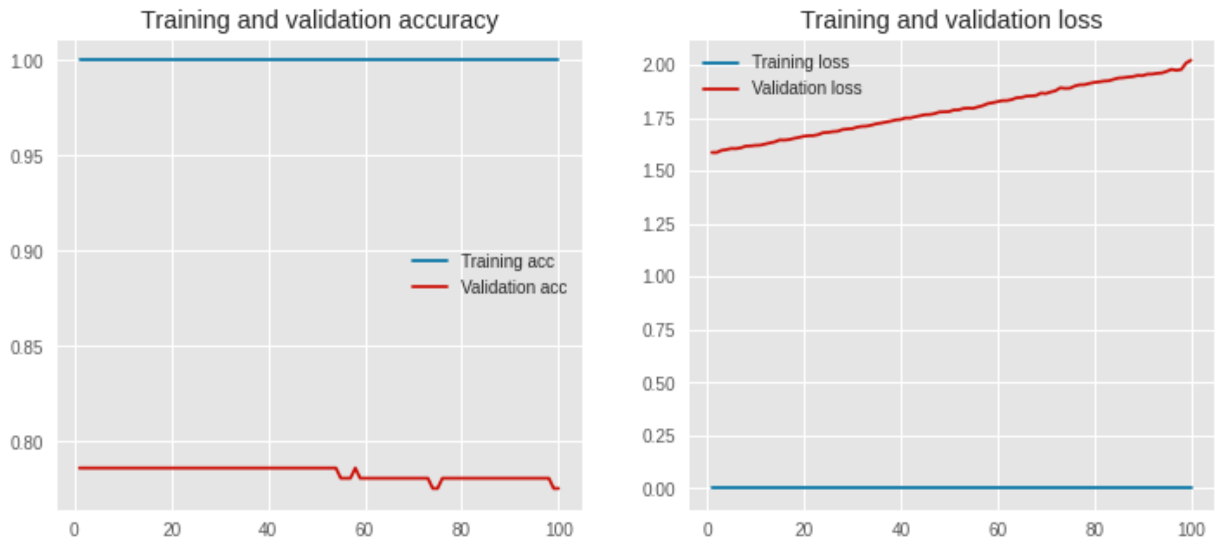
```
Training Accuracy: 1.0000
Testing Accuracy:  0.7754
```

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()
```

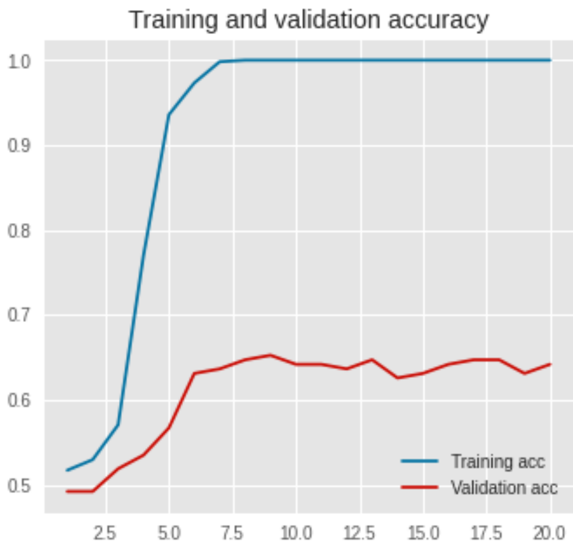
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 50)           128750

 flatten (Flatten)           (None, 5000)              0

 dense (Dense)               (None, 10)                50010

 dense_1 (Dense)             (None, 1)                 11

=================================================================
```

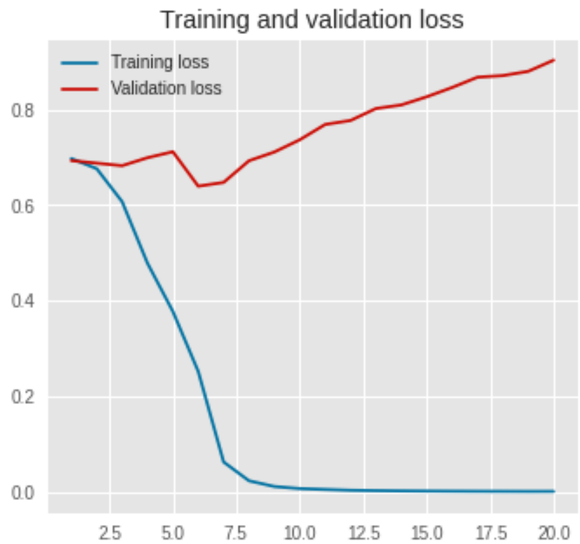
```
Total params: 178,771
Trainable params: 178,771
Non-trainable params: 0
```

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()
```

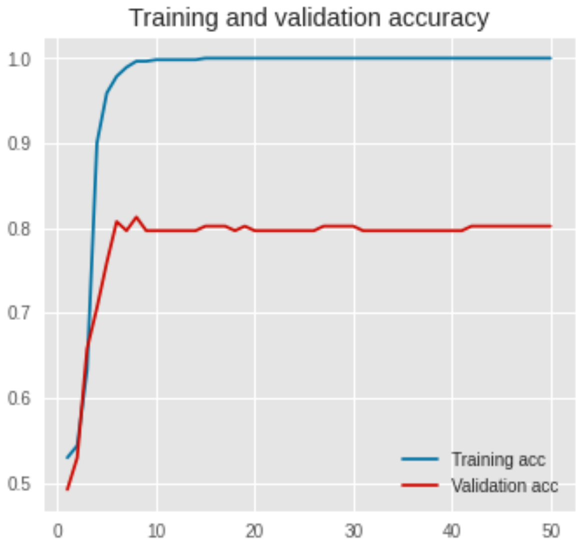
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 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]:

```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: 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()
```

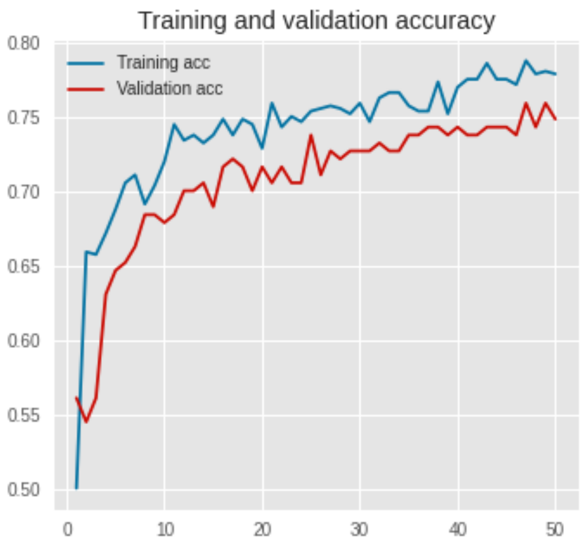
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 100, 50)           128750

 global_max_pooling1d_2 (Glo  (None, 50)               0
 balMaxPooling1D)

 dense_6 (Dense)             (None, 10)                510

 dense_7 (Dense)             (None, 1)                 11

=================================================================
```

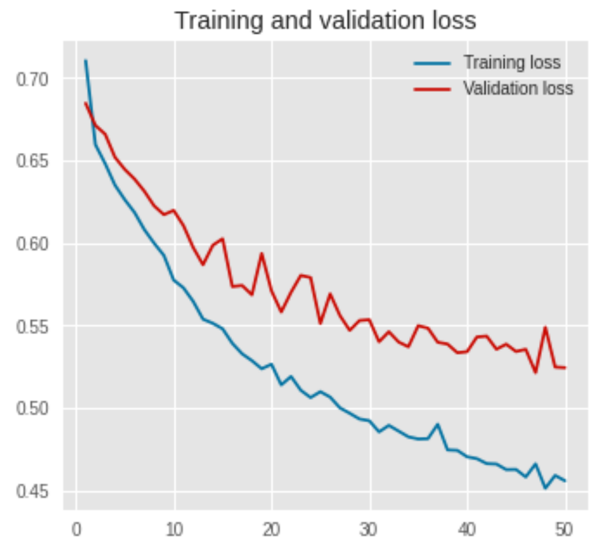
```
Total params: 129,271
Trainable params: 521
Non-trainable params: 128,750
```

In [54]:

```
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]:

```
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()
```

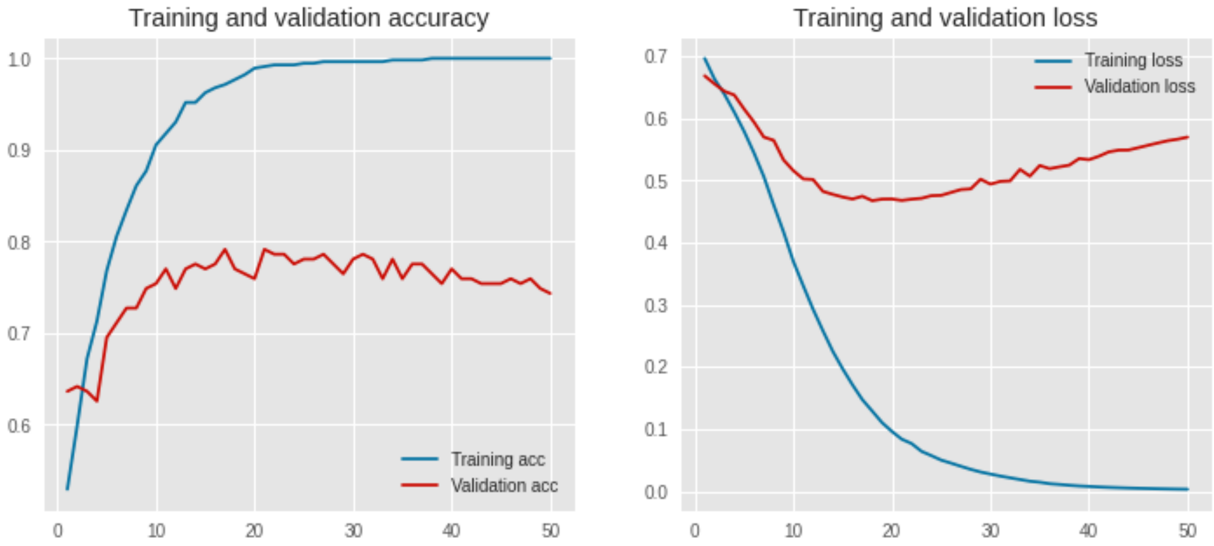
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 100, 50)           128750

 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()
```

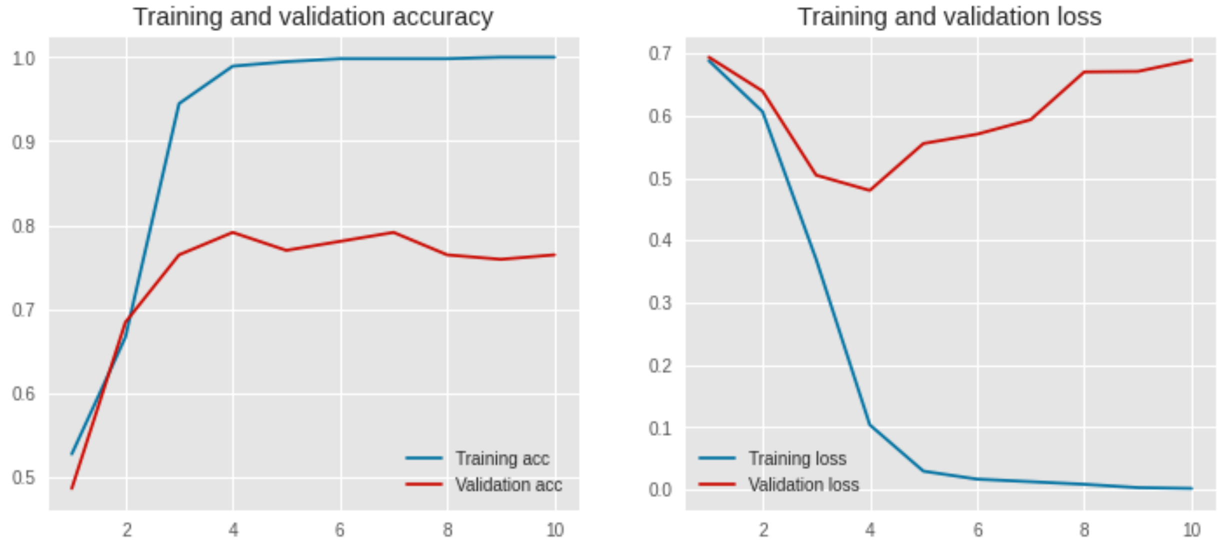
```
Model: "sequential_5"
_________________________________________________________________
```

```
 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]:

```python
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
```