In [ ]:

```
'''
LAB EVALUATION

Varun Kumar S   01FB15ECS338
Varun V         01FB15ECS341
Varun Y Vora    01FB15ECS342
Vishwas Satish  01FB15ECS355



Approach taken:

We made use of glove word embedding to convert the given words to a 100 dim
ensional vector.
The words were taken by concatenating summary and text and clipping the res
ulting text to just 50 words.
We used the rating to act as the label for a given review, 0 being
negative and 1 being positive.
We randomly chose 1500 entries from both positive and negative review data
sets to form the training data
and 150 entries from each to form the testing data.

Each sentence was represented as a list and the words of the sentence
forming the list elements,
so the review was a multidimensional list. This was converted to its
respective word vector with glove.

The RNN we used consisted of an embedding layer, an LSTM layer a fully conn
ected layer and an output layer.
This word vector was passed as an input for the created RNN. The hyper
parameters were as follows:

Dropout: 0.5
Activation function: Sigmoid
Error function: Binary Cross Entropy
Number of epochs: 16

Result:

Loss: 0.263
Accuracy: 89.26 %
Precision: 0.9306
Recall: 0.9489
'''
```

In [4]:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import json
import numpy as np
import keras.backend as K
from keras.utils import to_categorical
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
```

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


# 1. Loading the data
print("loading data...")

pos_file_name = "pos_amazon_cell_phone_reviews.json"
neg_file_name = "neg_amazon_cell_phone_reviews.json"
pos_file = open(pos_file_name, "r")
neg_file = open(neg_file_name, "r")
pos_data = json.loads(pos_file.read())['root']
neg_data = json.loads(neg_file.read())['root']
print("Posititve data loaded. ", len(pos_data), "entries")
print("Negative data loaded. ", len(neg_data), "entries")

print("done loading data...")

plabels = []
nlabels = []

# 2.Process reviews into sentences
pos_sentences, neg_sentences = [], []
for entry in pos_data :
    pos_sentences.append(entry['summary'] + " . " + entry['text'])
    plabels.append(1)
for entry in neg_data :
    nlabels.append(0)
    neg_sentences.append(entry['summary'] + " . " + entry['text'])
print(len(pos_sentences))
print(len(neg_sentences))

texts = pos_sentences + neg_sentences
labels = [1]*len(pos_sentences) + [0]*len(neg_sentences)


# 3. Tokenize
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

MAX_SEQUENCE_LENGTH = 50

data = sequence.pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

labels = np.array(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

rest_data = data[3000:]
rest_labels = labels[3000:]
```

```
data = data[:3000]
labels = labels[:3000]

VALIDATION_SPLIT = 0.2
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]

print(len(x_train), len(y_train))

#4. Get embeddings using GloVe
embeddings_index = {}
f = open('glove.6B/glove.6B.50d.txt', 'r', encoding = 'utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))

EMBEDDING_DIM = MAX_SEQUENCE_LENGTH

embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector


from keras.layers import Embedding

embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)

def precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


#5. Designing and Training the LSTM model

batch_size = 128
model = Sequential()
```

```
model = Sequential()
model.add(embedding_layer)
model.add(LSTM(64))
model.add(Dropout(0.50))
model.add(Dense(1, activation='sigmoid'))


model.compile('adam', 'binary_crossentropy', metrics=['accuracy', precision
, recall])

print('Train...')

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=16,
          validation_data=[x_val, y_val])

#6. Reporting results
x = model.evaluate(rest_data[:5000], rest_labels[:5000])
print("Loss: ", x[0])
print("Accuracy: ", x[1])
print("Precision: ", x[2])
print("Recall: ", x[3])
```

```
Using TensorFlow backend.
```

```
loading data...
Posititve data loaded.  108664 entries
Negative data loaded.  13279 entries
done loading data...
108664
13279
Found 69671 unique tokens.
Shape of data tensor: (121943, 50)
Shape of label tensor: (121943,)
(3000, 50) (3000,) 600
[1 1 1 ..., 1 1 1]
2400 2400
Found 400000 word vectors.
Train...
Train on 2400 samples, validate on 600 samples
Epoch 1/16
2400/2400 [==============================] - 3s - loss: 0.4342 - acc: 0.863
3 - precision: 0.8833 - recall: 0.9766 - val_loss: 0.3485 - val_acc: 0.8933
- val_precision: 0.8948 - val_recall: 0.9982
Epoch 2/16
2400/2400 [==============================] - 2s - loss: 0.3652 - acc: 0.883
8 - precision: 0.8838 - recall: 1.0000 - val_loss: 0.3388 - val_acc: 0.8933
- val_precision: 0.8948 - val_recall: 0.9982
Epoch 3/16
2400/2400 [==============================] - 3s - loss: 0.3612 - acc: 0.883
8 - precision: 0.8838 - recall: 1.0000 - val_loss: 0.3330 - val_acc: 0.8933
- val_precision: 0.8948 - val_recall: 0.9982
Epoch 4/16
2400/2400 [==============================] - 3s - loss: 0.3545 - acc: 0.883
8 - precision: 0.8838 - recall: 1.0000 - val_loss: 0.3277 - val_acc: 0.8933
- val_precision: 0.8948 - val_recall: 0.9982
Epoch 5/16
2400/2400 [==============================] - 2s - loss: 0.3360 - acc: 0.883
3 - precision: 0.8837 - recall: 0.9995 - val_loss: 0.3073 - val_acc: 0.8933
- val_precision: 0.8948 - val_recall: 0.9982
```

```
Epoch 6/16
2400/2400 [==============================] - 2s - loss: 0.3159 - acc: 0.885
0 - precision: 0.8852 - recall: 0.9995 - val_loss: 0.2968 - val_acc: 0.8933
- val_precision: 0.8975 - val_recall: 0.9944
Epoch 7/16
2400/2400 [==============================] - 2s - loss: 0.2959 - acc: 0.887
1 - precision: 0.8924 - recall: 0.9919 - val_loss: 0.2665 - val_acc: 0.8900
- val_precision: 0.8944 - val_recall: 0.9944
Epoch 8/16
2400/2400 [==============================] - 2s - loss: 0.2793 - acc: 0.890
0 - precision: 0.8952 - recall: 0.9921 - val_loss: 0.2548 - val_acc: 0.8950
- val_precision: 0.9142 - val_recall: 0.9738
Epoch 9/16
2400/2400 [==============================] - 2s - loss: 0.2799 - acc: 0.895
0 - precision: 0.9069 - recall: 0.9827 - val_loss: 0.2480 - val_acc: 0.9033
- val_precision: 0.9092 - val_recall: 0.9904
Epoch 10/16
2400/2400 [==============================] - 2s - loss: 0.2617 - acc: 0.899
6 - precision: 0.9109 - recall: 0.9825 - val_loss: 0.2433 - val_acc: 0.8917
- val_precision: 0.9228 - val_recall: 0.9588
Epoch 11/16
2400/2400 [==============================] - 2s - loss: 0.2595 - acc: 0.895
4 - precision: 0.9131 - recall: 0.9748 - val_loss: 0.2622 - val_acc: 0.9033
- val_precision: 0.9051 - val_recall: 0.9962
Epoch 12/16
2400/2400 [==============================] - 2s - loss: 0.2454 - acc: 0.900
4 - precision: 0.9145 - recall: 0.9793 - val_loss: 0.2381 - val_acc: 0.8967
- val_precision: 0.9325 - val_recall: 0.9533
Epoch 13/16
2400/2400 [==============================] - 2s - loss: 0.2319 - acc: 0.910
4 - precision: 0.9214 - recall: 0.9826 - val_loss: 0.2336 - val_acc: 0.9083
- val_precision: 0.9254 - val_recall: 0.9755
Epoch 14/16
2400/2400 [==============================] - 2s - loss: 0.2206 - acc: 0.911
2 - precision: 0.9296 - recall: 0.9731 - val_loss: 0.2228 - val_acc: 0.9117
- val_precision: 0.9289 - val_recall: 0.9756
Epoch 15/16
2400/2400 [==============================] - 2s - loss: 0.2166 - acc: 0.915
4 - precision: 0.9338 - recall: 0.9737 - val_loss: 0.2254 - val_acc: 0.9050
- val_precision: 0.9395 - val_recall: 0.9551
Epoch 16/16
2400/2400 [==============================] - 2s - loss: 0.2092 - acc: 0.917
9 - precision: 0.9367 - recall: 0.9731 - val_loss: 0.2182 - val_acc: 0.8950
- val_precision: 0.9420 - val_recall: 0.9403
4960/5000 [============================>.] - ETA: 0sLoss:  0.263203605509
Accuracy:  0.8926
Precision:  0.930600106049
Recall:  0.948868327332
```