lstm_toxic_single+multiple_final

December 5, 2019

In [1]:
```python
# import necessary libraries
import pandas as pd
import numpy as np
from numpy import array
from numpy import asarray
from numpy import zeros
import pickle

import nltk
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
en_stop = set(nltk.corpus.stopwords.words('english'))

from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, LSTM
from keras.layers import GlobalMaxPooling1D
from keras.models import Model
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.layers import Input
from keras.layers.merge import Concatenate
from keras import backend as K
from keras.models import load_model

import re
import matplotlib.pyplot as plt
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
```

```
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

```
Using TensorFlow backend.
```

In [0]:
```python
# Read in the dataset
toxic_comments = pd.read_csv("toxic_comments_cleaned_df.csv")
```

In [3]:
```python
print(toxic_comments.shape)

toxic_comments.head()
```

```
(159571, 9)
```

Out[3]:
```
                 id  ...                               clean_comment_text
0  0000997932d777bf  ...  explanation edits make username hardcore metal...
1  000103f0d9cfb60f  ...  aww match background colour seemingly stuck th...
2  000113f07ec002fd  ...  hey man really try edit war guy constantly rem...
3  0001b41b1c6bb37e  ...  make real suggestion improvement wonder sectio...
4  0001d958c54c6e35  ...                      sir hero chance remember page

[5 rows x 9 columns]
```

In [0]:
```python
# remove any records where any row contains a null value or empty string
filter = toxic_comments["comment_text"] != ""
toxic_comments = toxic_comments[filter]
toxic_comments = toxic_comments.dropna()
```

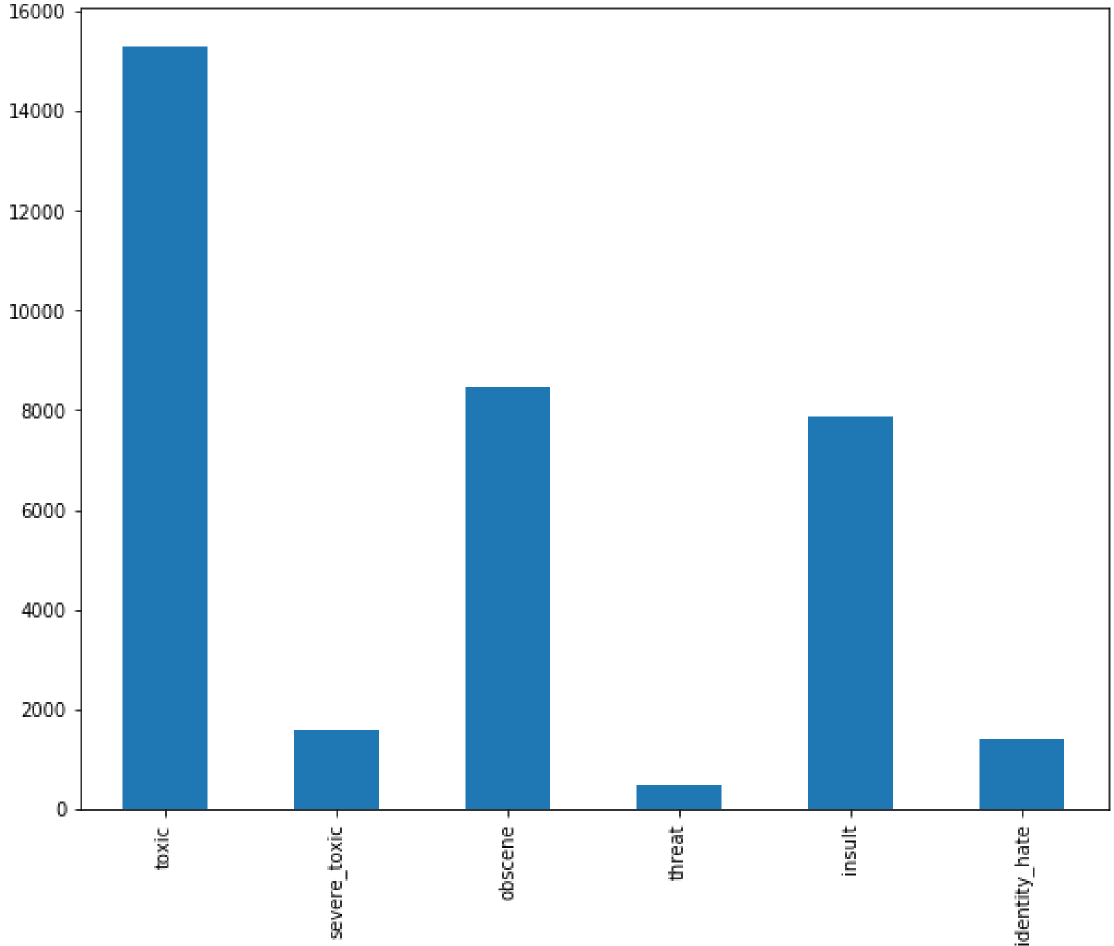
In [5]:
```python
toxic_comments.shape
```

Out[5]:
```
(159548, 9)
```

In [0]:
```python
# Look at the distribution of labels
def plot_label_frequency(df, label_columns):
    toxic_comments_labels = df[label_columns]
    fig_size = plt.rcParams["figure.figsize"]
    fig_size[0] = 10
    fig_size[1] = 8
    plt.rcParams["figure.figsize"] = fig_size
    bar_plot = toxic_comments_labels.sum(axis=0).plot.bar()
    return bar_plot
```

In [8]:
```python
label_columns = ["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hat
plot_label_frequency(toxic_comments, label_columns)
```

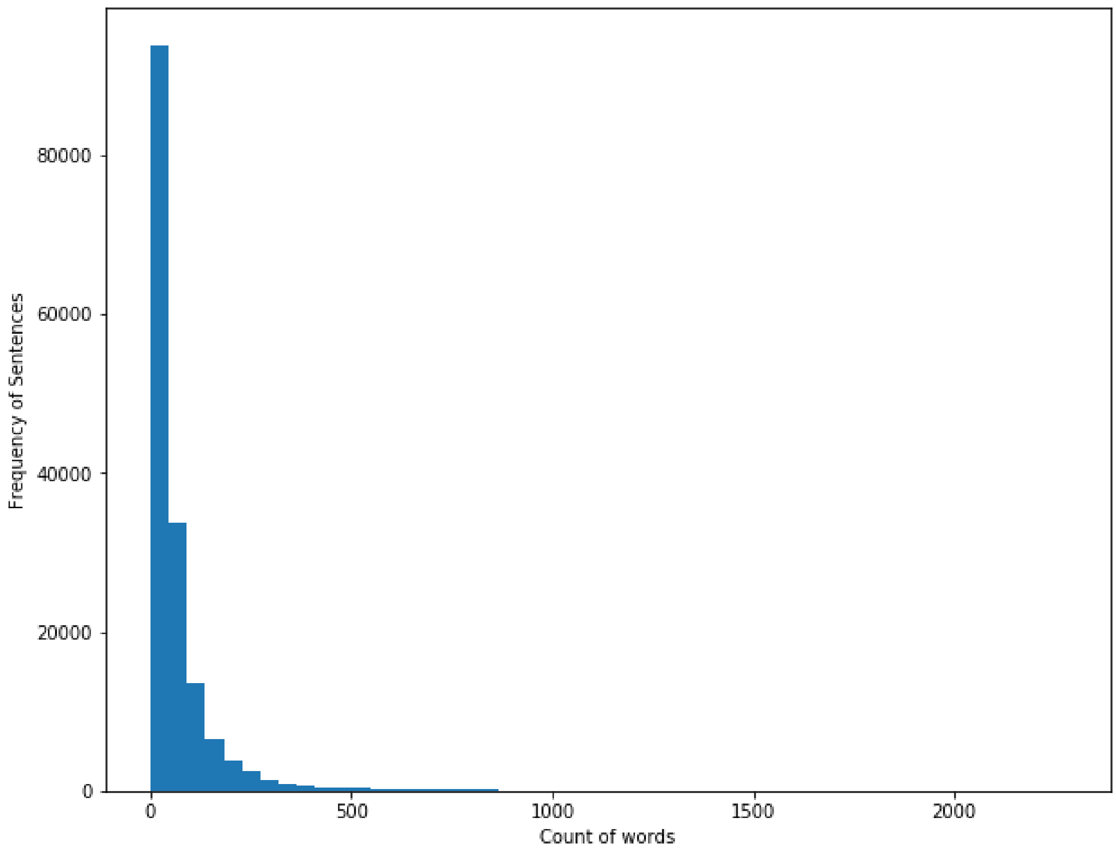
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f123e6cdcc0>




```
In [0]: # Create input set(feature variables) -  clean all the comments and store them in a va
        X = []
        sentences = list(toxic_comments["clean_comment_text"])
        for sen in sentences:
            X.append(sen)
        # X = pd.read_csv("X_toxic_feature_google_col.csv")["columnn"].tolist()
        # Create output set (target/labels)
        y = toxic_comments[label_columns].values
```

```
In [10]: sentences = list(toxic_comments["comment_text"])
         sentence_list = [ sen.split(' ') for sen in sentences]
         plt.hist([len(s) for s in sentence_list], bins=50)
         plt.xlabel('Count of words')
         plt.ylabel('Frequency of Sentences')

         plt.show()
```




```
In [0]: # Split it into train and test sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=
```

```
In [0]: X_train1 = list(str(elem) for elem in X_train)
        X_test1 = list(str(elem) for elem in X_test)
        # y_train1 = list(str(elem) for elem in y_train)
        # y_test1 = list(str(elem) for elem in y_test)
```

```
In [0]: tokenizer_toxic = Tokenizer(num_words=5000)
        tokenizer_toxic.fit_on_texts(X_train1)

        # saving
        with open('tokenizer_toxic_final.pickle', 'wb') as handle:
            pickle.dump(tokenizer_toxic, handle, protocol=pickle.HIGHEST_PROTOCOL)

        # loading
        with open('tokenizer_toxic_final.pickle', 'rb') as handle:
            tokenizer_toxic = pickle.load(handle)
```

```
In [0]: X_train1 = tokenizer_toxic.texts_to_sequences(X_train1)
        X_test1 = tokenizer_toxic.texts_to_sequences(X_test1)
```

```
vocab_size = len(tokenizer_toxic.word_index) + 1

maxlen = 200

X_train1 = pad_sequences(X_train1, padding='post', maxlen=maxlen)
X_test1 = pad_sequences(X_test1, padding='post', maxlen=maxlen)
```

In [0]:
```
# Define helper functions to get pre-trained glove word vector embeddings
# and create an embeddings matrix

def get_word_embeddings():
    embeddings_dictionary = dict()
    glove_file = open('glove.6B.50d.txt', encoding="utf8")

    for line in glove_file:
        records = line.split()
        word = records[0]
        vector_dimensions = asarray(records[1:], dtype='float32')
        embeddings_dictionary[word] = vector_dimensions

    glove_file.close()
    return embeddings_dictionary

embeddings_dictionary = get_word_embeddings()

def get_embedding_matrix():
    embedding_matrix = zeros((vocab_size, 50))
    for word, index in tokenizer_toxic.word_index.items():
        embedding_vector = embeddings_dictionary.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector
    return embedding_matrix

embedding_matrix = get_embedding_matrix()
```

In [16]:
```
embedding_matrix.shape
```

Out[16]:
```
(154189, 50)
```

In [0]:
```
embedding_matrix.shape
```

Out[0]:
```
(154297, 100)
```

In [0]:
```
# Define functions to be able to calculate additional metrics like precision,recall, f

def recall_m(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

    def precision_m(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 f1_m(y_true, y_pred):
        precision = precision_m(y_true, y_pred)
        recall = recall_m(y_true, y_pred)
        return 2*((precision*recall)/(precision+recall+K.epsilon()))
    def hamming_loss(y_true, y_pred):
      return K.mean(y_true*(1-y_pred)+(1-y_true)*y_pred)
```

In [18]:
```
# Approach 1
# Use a single dense layer with six outputs with sigmoid activation functions and bin
# Each neuron in the output dense layer will represent one of the six output labels.
from tensorflow import set_random_seed
set_random_seed(1)
deep_inputs_single = Input(shape=(maxlen,))
embedding_layer_single = Embedding(vocab_size, 50, weights=[embedding_matrix], trainal
LSTM_Layer_1_single = LSTM(128)(embedding_layer_single)
dense_layer_1_single = Dense(6, activation='sigmoid')(LSTM_Layer_1_single)
model_toxic_single = Model(inputs=deep_inputs_single, outputs=dense_layer_1_single)

model_toxic_single.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The nar
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/nn_i
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
In [19]: print(model_toxic_single.summary())
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 200)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 200, 50)           7709450
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               91648
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 7,801,872
Trainable params: 92,422
Non-trainable params: 7,709,450
_________________________________________________________________
None
```

```
In [0]: from keras.utils import plot_model
        plot_model(model, to_file='model_plot4a.png', show_shapes=True, show_layer_names=True)
```

```
In [20]: history_toxic_single = model_toxic_single.fit(X_train1, y_train, batch_size=128, epoch
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

Train on 102110 samples, validate on 25528 samples
Epoch 1/5
102110/102110 [==============================] - 431s 4ms/step - loss: 0.1487 - acc: 0.9634 - 
Epoch 2/5
102110/102110 [==============================] - 429s 4ms/step - loss: 0.0909 - acc: 0.9710 - 
Epoch 3/5
102110/102110 [==============================] - 444s 4ms/step - loss: 0.0667 - acc: 0.9777 - 
Epoch 4/5
102110/102110 [==============================] - 445s 4ms/step - loss: 0.0616 - acc: 0.9789 - 
Epoch 5/5
102110/102110 [==============================] - 446s 4ms/step - loss: 0.0690 - acc: 0.9783 - 
```

```
In [0]: model_toxic_single.save("toxic_lstm_single_5.h5")
```

```
In [0]: # load model from single file
        toxic_lstm_single_5 = load_model('toxic_lstm_single_5.h5', custom_objects={'f1_m': f1_m
                                                                                     'hamming_los
```

```
In [22]: loss, accuracy, f1_score, precision, recall, hamming = model_toxic_single.evaluate(X_t

         print("Test Score:", loss)
         print("Test Accuracy:", accuracy)
         print("Test Precision:", precision)
         print("Test Recall:", recall)
         print("Test F1-score:", f1_score)
         print("Test hamming_loss:", hamming)
```

```
31910/31910 [==============================] - 47s 1ms/step
Test Score: 0.09585073696707987
Test Accuracy: 0.9752742002242712
Test Precision: 0.5942442163857723
Test Recall: 0.3357364522144362
Test F1-score: 0.4060554962552703
Test hamming_loss: 0.04034598560630808
```

```
In [23]: model_toxic_single.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc
         history_toxic_single_10 = model_toxic_single.fit(X_train1, y_train, batch_size=128, ep
```

```
Train on 102110 samples, validate on 25528 samples
Epoch 1/10
102110/102110 [==============================] - 450s 4ms/step - loss: 0.0694 - acc: 0.9780 - 
Epoch 2/10
102110/102110 [==============================] - 448s 4ms/step - loss: 0.0657 - acc: 0.9789 - 
Epoch 3/10
102110/102110 [==============================] - 451s 4ms/step - loss: 0.0601 - acc: 0.9792 - 
Epoch 4/10
102110/102110 [==============================] - 448s 4ms/step - loss: 0.0591 - acc: 0.9796 - 
Epoch 5/10
102110/102110 [==============================] - 449s 4ms/step - loss: 0.0563 - acc: 0.9804 - 
Epoch 6/10
102110/102110 [==============================] - 450s 4ms/step - loss: 0.0551 - acc: 0.9806 - 
Epoch 7/10
102110/102110 [==============================] - 447s 4ms/step - loss: 0.0537 - acc: 0.9811 - 
Epoch 8/10
102110/102110 [==============================] - 445s 4ms/step - loss: 0.0528 - acc: 0.9812 - 
Epoch 9/10
102110/102110 [==============================] - 447s 4ms/step - loss: 0.0515 - acc: 0.9815 - 
Epoch 10/10
102110/102110 [==============================] - 447s 4ms/step - loss: 0.0502 - acc: 0.9818 - 
```

In [0]:
```
model_toxic_single.save("toxic_lstm_single_10.h5")
```

In [0]:
```
# load model from single file
toxic_lstm_single_10 = load_model('toxic_lstm_single_10.h5', custom_objects={'f1_m': f
```

In [24]:
```
loss, accuracy, f1_score, precision, recall, hamming = model_toxic_single.evaluate(X_

 print("Test Score:", loss)
 print("Test Accuracy:", accuracy)
 print("Test Precision:", precision)
 print("Test Recall:", recall)
 print("Test F1-score:", f1_score)
 print("Test hamming_loss:", hamming)
```

```
31910/31910 [==============================] - 46s 1ms/step
Test Score: 0.05413343073504407
Test Accuracy: 0.9812702265625574
Test Precision: 0.7285210935645849
Test Recall: 0.5669637860642149
Test F1-score: 0.6119952032835795
Test hamming_loss: 0.027499208569179943
```

In [0]:
```
# Approach 1 with ACTIVATION SOFTMAX
# Use a single dense layer with six outputs with sigmoid activation functions and bina
# Each neuron in the output dense layer will represent one of the six output labels.
from tensorflow import set_random_seed
set_random_seed(1)
deep_inputs_single = Input(shape=(maxlen,))
embedding_layer_single = Embedding(vocab_size, 50, weights=[embedding_matrix], trainab
LSTM_Layer_1_single = LSTM(128)(embedding_layer_single)
dense_layer_1_single = Dense(6, activation='softmax')(LSTM_Layer_1_single)
model_toxic_single_soft = Model(inputs=deep_inputs_single, outputs=dense_layer_1_singl

model_toxic_single_soft.compile(loss='binary_crossentropy', optimizer='adam', metrics=
```

In [27]:
```
history_toxic_single_epoch = model_toxic_single_soft.fit(X_train1, y_train, batch_size
```

```
Train on 102110 samples, validate on 25528 samples
Epoch 1/5
102110/102110 [==============================] - 432s 4ms/step - loss: 0.2312 - acc: 0.9634 - 
Epoch 2/5
102110/102110 [==============================] - 447s 4ms/step - loss: 0.2309 - acc: 0.9635 - 
Epoch 3/5
102110/102110 [==============================] - 445s 4ms/step - loss: 0.2309 - acc: 0.9635 - 
Epoch 4/5
102110/102110 [==============================] - 445s 4ms/step - loss: 0.2308 - acc: 0.9635 - 
Epoch 5/5
102110/102110 [==============================] - 445s 4ms/step - loss: 0.2307 - acc: 0.9635 - 
```

In [28]:
```
loss, accuracy, f1_score, precision, recall, hamming = model_toxic_single_soft.evalua

print("Test Score:", loss)
print("Test Accuracy:", accuracy)
print("Test Precision:", precision)
print("Test Recall:", recall)
print("Test F1-score:", f1_score)
print("Test hamming_loss:", hamming)
```

```
31910/31910 [==============================] - 46s 1ms/step
Test Score: 0.22980441345629815
Test Accuracy: 0.964081265058909
Test Precision: 0.007019742190445602
Test Recall: 0.0008332960510732088
Test F1-score: 0.0014671716149120039
Test hamming_loss: 0.18503734519380044
```

In [0]:
```
# saving
with open('tokenizer_toxic.pickle', 'wb') as handle:
    pickle.dump(tokenizer_toxic, handle, protocol=pickle.HIGHEST_PROTOCOL)

# loading
with open('tokenizer_toxic.pickle', 'rb') as handle:
    tokenizer_toxic = pickle.load(handle)
```

In [0]:
```
# Approach 2
# Create one dense output layer for each label.
# Total of 6 dense layers in the output. Each layer will have its own sigmoid function
```

In [0]:
```
layers_dict = {}
for i in range(len(label_columns)):
  layers_dict["y"+str(i+1)+"_train"] = y_train[:,[i]]
  layers_dict["y"+str(i+1)+"_test"] = y_test[:,[i]]
```

In [0]:
```
layers_dict["y1_train"]
```

Out[0]:
```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [0],
       [0]])
```

In [0]:
```
input_1 = Input(shape=(maxlen,))
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], trainable=Fal
LSTM_Layer1 = LSTM(128)(embedding_layer)
```

```
output1 = Dense(1, activation='sigmoid')(LSTM_Layer1)
output2 = Dense(1, activation='sigmoid')(LSTM_Layer1)
output3 = Dense(1, activation='sigmoid')(LSTM_Layer1)
output4 = Dense(1, activation='sigmoid')(LSTM_Layer1)
output5 = Dense(1, activation='sigmoid')(LSTM_Layer1)
output6 = Dense(1, activation='sigmoid')(LSTM_Layer1)

model = Model(inputs=input_1, outputs=[output1, output2, output3, output4, output5, ou
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc', f1_m,preci
```

```
In [0]: print(model.summary())
```

```
Model: "model_4"
____________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
============================================================================================
input_4 (InputLayer)            (None, 200)          0
____________________________________________________________________________________________
embedding_4 (Embedding)         (None, 200, 100)     15429700    input_4[0][0]
____________________________________________________________________________________________
lstm_4 (LSTM)                   (None, 128)          117248      embedding_4[0][0]
____________________________________________________________________________________________
dense_9 (Dense)                 (None, 1)            129         lstm_4[0][0]
____________________________________________________________________________________________
dense_10 (Dense)                (None, 1)            129         lstm_4[0][0]
____________________________________________________________________________________________
dense_11 (Dense)                (None, 1)            129         lstm_4[0][0]
____________________________________________________________________________________________
dense_12 (Dense)                (None, 1)            129         lstm_4[0][0]
____________________________________________________________________________________________
dense_13 (Dense)                (None, 1)            129         lstm_4[0][0]
____________________________________________________________________________________________
dense_14 (Dense)                (None, 1)            129         lstm_4[0][0]
============================================================================================
Total params: 15,547,722
Trainable params: 118,022
Non-trainable params: 15,429,700
____________________________________________________________________________________________
None
```

```
In [0]: history = model.fit(x=X_train1, y=[layers_dict["y1_train"],
                                           layers_dict["y2_train"],
                                           layers_dict["y3_train"],
                                           layers_dict["y4_train"],
                                           layers_dict["y5_train"],
                                           layers_dict["y6_train"]], batch_size=8192, epochs=5
```

```
Train on 102124 samples, validate on 25532 samples
Epoch 1/5
```

```
102124/102124 [==============================] - 22s 219us/step - loss: 3.5549 - dense_9_loss:
Epoch 2/5
102124/102124 [==============================] - 19s 187us/step - loss: 0.9005 - dense_9_loss:
Epoch 3/5
102124/102124 [==============================] - 19s 185us/step - loss: 0.8565 - dense_9_loss:
Epoch 4/5
102124/102124 [==============================] - 19s 185us/step - loss: 0.8519 - dense_9_loss:
Epoch 5/5
102124/102124 [==============================] - 19s 185us/step - loss: 0.8495 - dense_9_loss:
```

```
In [0]: eval = model.evaluate(x=X_test1, y=[layers_dict["y1_test"],
                                           layers_dict["y2_test"],
                                           layers_dict["y3_test"],
                                           layers_dict["y4_test"],
                                           layers_dict["y5_test"],
                                           layers_dict["y6_test"]], verbose=1)

        # print("Test Score:", loss)
        # print("Test Accuracy:", accuracy)
        # print("Test Precision:", precision)
        # print("Test Recall:", recall)
        # print("Test F1-score:", f1_score)
```

```
31915/31915 [==============================] - 107s 3ms/step
```

```
Out[0]: [0.8436724258257307,
         0.3169314009538549,
         0.05507167037658145,
         0.20436195503501856,
         0.02171659932050725,
         0.19646163133829342,
         0.04912915760060309,
         0.9034936550286203,
         0.005817833914944282,
         0.013034621664074434,
         0.004087046614713697,
         0.9898793670687764,
         0.0016711052618634825,
         0.0020053264098576053,
         0.0015039948671565953,
         0.9478301739087583,
         0.005147004400172914,
         0.008021305639430421,
         0.004177763632765836,
         0.9964593451355163,
         0.0,
```

```
0.0,
0.0,
0.9505248315932328,
0.005882290764398827,
0.010026632049288026,
0.004428429583406516,
0.9911953626821244,
0.0010026631451654114,
0.0010026632049288027,
0.0010026632049288027]
```

```
In [0]:
```