```
# import necessary libraries
import pandas as pd
import numpy as np
from numpy import array
from numpy import asarray
from numpy import zeros

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

import re
import pickle
import matplotlib.pyplot as plt

from keras import backend as K
from keras.models import load_model
```

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

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tens`

```
# read the data in from the csv
movies = pd.read_csv("movies_small_subset_df.csv")
movies = movies[['MovieID', 'MovieName', 'Genre', 'Plot',
       'clean_plot_text']]
```

```
# Function for converting genre column to a list such that it can be indexed to get g
def format_list(x):
    x = x.replace("'","")
    x = x.replace("[","")
    x = x.replace("]","")
    x = x.split(',')
    result = []
    for word in x:
        result.append(word.strip())
    return result
movies["Genre"] = movies["Genre"].apply(lambda x : format_list(x))
movies.head(5)
```

	MovieID	MovieName	Genre	Plot	clean_plot_text
0	23890098	Taxi Blues	[Drama, World cinema]	Shlykov, a hard-working taxi driver and Lyosha...	shlykov hard work taxi driver lyosha saxophoni...
1	31186339	The Hunger Games	[Action, Drama]	The nation of Panem consists of a wealthy Capi...	nation panem consist wealthy capitol twelve po...
2	20663735	Narasimham	[Action,	Poovalli Induchoodan is	poovalli induchoodan sentence six year

```
movies.shape
```

```
(35523, 5)
```

```
# get a list of all unique genres from genres column in dataframe
unique_genre_list = list(set([a for b in movies.Genre.tolist() for a in b]))


for i in range(len(unique_genre_list)):
  movies[unique_genre_list[i]] = pd.Series([0 for x in range(len(movies.index))], ind

movies.head(2)
```

	MovieID	MovieName	Genre	Plot	clean_plot_text	Short Film	Comedy	Romance Film	T
0	23890098	Taxi Blues	[Drama, World cinema]	Shlykov, a hard-working taxi driver and Lyosha...	shlykov hard work taxi driver lyosha saxophoni...	0	0	0	

```
movies.shape
```

```
(35523, 13)
```

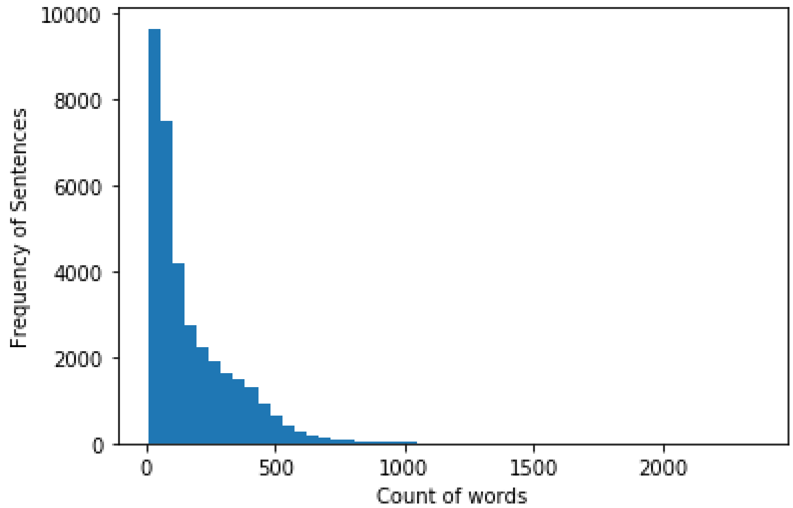
```
for gen in unique_genre_list:
  movies[gen] = movies["Genre"].apply(lambda x : (pd.Series([gen]).isin(x)).astype(in
movies.head(5)
```

	MovieID	MovieName	Genre	Plot	clean_plot_text	Short Film	Comedy	Roman Fi
0	23890098	Taxi Blues	[Drama, World cinema]	Shlykov, a hard-working taxi driver and Lyosha...	shlykov hard work taxi driver lyosha saxophoni...	0	0	
1	31186339	The Hunger Games	[Action, Drama]	The nation of Panem consists of a wealthy Capi...	nation panem consist wealthy capitol twelve po...	0	0	
2	20663735	Narasimham	[Action, Drama]	Poovalli Induchoodan is sentenced for six yea...	poovalli induchoodan sentence six year prison ...	0	0	
				The Lemon				

```
#movies.to_csv("movies_one_hot_df.csv")
#movies = pd.read_csv("movies_one_hot_df.csv")
```

```
sentences = list(movies["clean_plot_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()
```




```
X = []
for sen in sentences:
    X.append(sen)
# Create output set (target/labels)
y = movies[unique_genre_list].values


# 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_stat


X_train1 = list(str(elem) for elem in X_train)
X_test1 = list(str(elem) for elem in X_test)


tokenizer_movie = Tokenizer(num_words=5000)
tokenizer_movie.fit_on_texts(X_train1)

# saving
with open('tokenizer_movie_final.pickle', 'wb') as handle:
    pickle.dump(tokenizer_movie, handle, protocol=pickle.HIGHEST_PROTOCOL)
# loading
with open('tokenizer_movie_final.pickle', 'rb') as handle:
    tokenizer_movie = pickle.load(handle)




X_train1 = tokenizer_movie.texts_to_sequences(X_train1)
X_test1 = tokenizer_movie.texts_to_sequences(X_test1)

vocab_size = len(tokenizer_movie.word_index) + 1

maxlen = 500

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

```
# 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.100d.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, 100))
    for word, index in tokenizer_movie.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()


# Define functions to be able to calculate additional metrics like precision,recall,

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)
```

```
# 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.
# ACTIVATION : SIGMOID
from tensorflow import set_random_seed
set_random_seed(1)
deep_inputs_single = Input(shape=(maxlen,))
embedding_layer_single = Embedding(vocab_size, 100, weights=[embedding_matrix], train
LSTM_Layer_1_single = LSTM(128)(embedding_layer_single)
dense_layer_1_single = Dense(8, activation='sigmoid')(LSTM_Layer_1_single)
model_movie_single = Model(inputs=deep_inputs_single, outputs=dense_layer_1_single)

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

print(model_movie_single.summary())
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 500)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 500, 100)          10465300
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               117248
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 1032
=================================================================
Total params: 10,583,580
Trainable params: 118,280
Non-trainable params: 10,465,300
_________________________________________________________________
None
```

```
history_movie_single_5 = model_movie_single.fit(X_train1, y_train, batch_size=128, ep
```

```
Train on 22734 samples, validate on 5684 samples
Epoch 1/5
22734/22734 [==============================] - 134s 6ms/step - loss: 0.4890 - acc
Epoch 2/5
22734/22734 [==============================] - 132s 6ms/step - loss: 0.4746 - acc
Epoch 3/5
22734/22734 [==============================] - 132s 6ms/step - loss: 0.4740 - acc
Epoch 4/5
22734/22734 [==============================] - 131s 6ms/step - loss: 0.4740 - acc
Epoch 5/5
22734/22734 [==============================] - 131s 6ms/step - loss: 0.4750 - acc
```

```
loss, accuracy, f1_score, precision, recall, hamming = model_movie_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)
```

```
7105/7105 [==============================] - 52s 7ms/step
Test Score: 0.4734555327162451
Test Accuracy: 0.7906052076002815
Test Precision: 0.5420741732904057
Test Recall: 0.3094173200989508
Test F1-score: 0.3932593436747852
Test hamming_loss: 0.3098531847642728
```

```
model_movie_single.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
history_movie_single_10 = model_movie_single.fit(X_train1, y_train, batch_size=128, e
```

```
Train on 22734 samples, validate on 5684 samples
Epoch 1/10
22734/22734 [==============================] - 134s 6ms/step - loss: 0.4735 - acc
Epoch 2/10
22734/22734 [==============================] - 132s 6ms/step - loss: 0.4729 - acc
Epoch 3/10
22734/22734 [==============================] - 132s 6ms/step - loss: 0.4728 - acc
Epoch 4/10
22734/22734 [==============================] - 132s 6ms/step - loss: 0.4730 - acc
Epoch 5/10
22734/22734 [==============================] - 132s 6ms/step - loss: 0.4579 - acc
Epoch 6/10
22734/22734 [==============================] - 131s 6ms/step - loss: 0.4430 - acc
Epoch 7/10
22734/22734 [==============================] - 131s 6ms/step - loss: 0.4395 - acc
Epoch 8/10
22734/22734 [==============================] - 132s 6ms/step - loss: 0.4382 - acc
Epoch 9/10
22734/22734 [==============================] - 131s 6ms/step - loss: 0.4310 - acc
Epoch 10/10
22734/22734 [==============================] - 130s 6ms/step - loss: 0.4203 - acc
```

```
loss, accuracy, f1_score, precision, recall, hamming = model_movie_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)
```

```
7105/7105 [==============================] - 53s 7ms/step
Test Score: 0.4190679441699673
Test Accuracy: 0.811963406052076
Test Precision: 0.6141312826694861
Test Recall: 0.3926639343927143
Test F1-score: 0.47751756736210416
Test hamming_loss: 0.26848946765725834
```

```
model_movie_single.save("movie_lstm_single_10.h5")
```

```
# load model from single file
movie_lstm_single_10 = load_model('movie_lstm_single_10.h5', custom_objects={'f1_m':
```

```
# Approach 1 with softmax activation
# 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.
# ACTIVATION : SOFTMAX
from tensorflow import set_random_seed
set_random_seed(1)
deep_inputs_single = Input(shape=(maxlen,))
embedding_layer_single = Embedding(vocab_size, 100, weights=[embedding_matrix], train
LSTM_Layer_1_single = LSTM(128)(embedding_layer_single)
dense_layer_1_single = Dense(8, activation='softmax')(LSTM_Layer_1_single)
model_movie_single_soft = Model(inputs=deep_inputs_single, outputs=dense_layer_1_sing

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

```
history_movie_single_5_soft = model_movie_single_soft.fit(X_train1, y_train, batch_si
```

```
Train on 22734 samples, validate on 5684 samples
Epoch 1/5
22734/22734 [==============================] - 136s 6ms/step - loss: 0.5170 - acc
Epoch 2/5
22734/22734 [==============================] - 134s 6ms/step - loss: 0.5123 - acc
Epoch 3/5
22734/22734 [==============================] - 133s 6ms/step - loss: 0.5119 - acc
Epoch 4/5
22734/22734 [==============================] - 133s 6ms/step - loss: 0.5113 - acc
Epoch 5/5
22734/22734 [==============================] - 133s 6ms/step - loss: 0.5135 - acc
```

```
loss, accuracy, f1_score, precision, recall, hamming = model_movie_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)
```

```
7105/7105 [==============================] - 52s 7ms/step
Test Score: 0.511114438531434
Test Accuracy: 0.7802076002814919
Test Precision: 0.045038699768195266
Test Recall: 0.0007870505973539412
Test F1-score: 0.0015467913112197436
Test hamming_loss: 0.26534659909497343
```