PROJECT ON NATURAL LANGUAGE PROCESSING

By,

SARIKAA S
2018103058
21-05-21

Problem Statement:

To classify movies into genres based on the plot summary using word embeddings and neural networks and evaluate the model.Automatic movie genre tagging is used in recommendation systems.

Dataset:

- The dataset used is MPST: Movie Plot Synopses with Tags.
- The dataset contains IMDB id, title, plot synopsis, genres for the movies as columns..
- There are 14,828 movie data in total.
- There 71 different movie genres.

```
df = pd.read_csv('C:/Users/ACER/Downloads/movie/mpst_full_data.csv', delimiter=',')
nRow, nCol = df.shape
df.head(5)
```

	imdb_id	title	plot_synopsis	tags	split	synopsis_source
0	tt0057603	I tre volti della paura	Note: this synopsis is for the orginal Italian...	cult, horror, gothic, murder, atmospheric	train	imdb
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	violence	train	imdb
2	tt0033045	The Shop Around the Corner	Matuschek's, a gift store in Budapest, is the ...	romantic	test	imdb
3	tt0113862	Mr. Holland's Opus	Glenn Holland, not a morning person by anyone'...	inspiring, romantic, stupid, feel-good	train	imdb
4	tt0086250	Scarface	In May 1980, a Cuban man named Tony Montana (A...	cruelty, murder, dramatic, cult, violence, atm...	val	imdb

Modules:

- Data Cleaning
- Tokenization
- Word Embedding and Model creation
- Fitting and Evaluating the model

Model Summary:

Layers:

- One Embedding layer
- Two LSTM layer
- Two Dropout Layer
- One Dense layer(Since there are 71 genres the number of neurons in the output layer is 71)

2018103058_NLP Last Checkpoint: 17 minutes ago (autosaved)

View Insert Cell Kernel Widgets Help

Run Code

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1200, 50)          6109750
_________________________________________________________________
lstm (LSTM)                  (None, 1200, 128)         91648
_________________________________________________________________
dropout (Dropout)            (None, 1200, 128)         0
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                49408
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 71)                4615
=================================================================
Total params: 6,255,421
Trainable params: 6,255,421
Non-trainable params: 0
_________________________________________________________________
```

Data Cleaning:

```python
import re

def decontracted(phrase):
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

Removing links,punctuation,white spaces and uppercase to lowercase ,Contraction Replacement:

```
from tqdm import tqdm
preprocessed_synopsis = []

for text in df['plot_synopsis'].values:
    text = re.sub(r"http\S+", "", text)
    text = BeautifulSoup(text, 'lxml').get_text()
    text = decontracted(text)
    text = re.sub("\S*\d\S*", "", text).strip()
    text = re.sub('[^A-Za-z]+', ' ', text)

    text = ' '.join(e.lower() for e in text.split() if e.lower() not in stopwords)
    preprocessed_synopsis.append(text.strip())
df['preprocessed_plots']=preprocessed_synopsis
```

```
def remove_spaces(x):
    x=x.split(",")
    nospace=[]
    for item in x:
        item=item.lstrip()
        nospace.append(item)
    return (",").join(nospace)

df['tags']=df['tags'].apply(remove_spaces)
df['tags']
```

```
Out[6]: 0                    cult,horror,gothic,murder,atmospheric
        1                                                 violence
        2                                                 romantic
        3                       inspiring,romantic,stupid,feel-good
        4        cruelty,murder,dramatic,cult,violence,atmosphe...
                                       ...
        14823                                        comedy,murder
        14824                             good versus evil,violence
        14825                                             anti war
        14826                                               murder
        14827                                      christian film
        Name: tags, Length: 14828, dtype: object
```

Lemmatization and stemming:

```
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import word_tokenize
stemmer = PorterStemmer()
```

```
wnl = WordNetLemmatizer()
def stem_words(text):
    word_tokens = word_tokenize(text)
    stems = [stemmer.stem(word) for word in text]
    lemmatized_string = ' '.join([wnl.lemmatize(words) for words in stems])

    return lemmatized_string

for i in df['preprocessed_plots']:
    df['preprocessed_plots'][i]=stem_words(i)
```

Splitting the dataset into train and test sets:

```
train=df.loc[df.split=='train']
train=train.reset_index()
test=df.loc[df.split=='test']
test=test.reset_index()
```

Vectorizer:

In [8]:
```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(tokenizer = lambda x: x.split(","), binary='true')
y_train = vectorizer.fit_transform(train['tags']).toarray()
y_test = vectorizer.transform(test['tags']).toarray()

print(y_train)
```

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

In [9]:
```
vectorizer.inverse_transform(y_train[0])
```

Out[9]:
```
[array(['atmospheric', 'cult', 'gothic', 'horror', 'murder'], dtype='<U18')]
```

Tokenization and padding:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tensorflow import keras
from tensorflow.keras import layers
```

```
vect=Tokenizer()
vect.fit_on_texts(train['plot_synopsis'])
vocab_size = len(vect.word_index) + 1
print(vocab_size)
```

```
122195
```

```
xtrain1 = vect.texts_to_sequences(train['preprocessed_plots'])
max_length = vocab_size
xtrain = pad_sequences(xtrain1, maxlen=1200, padding='post')
print(xtrain)
```

```
[[  779  4660 62208 ...     0     0     0]
 [   51  4481   143 ...     0     0     0]
 [ 3063   429   188 ...    75   140  6946]
 ...
 [  140  2717   539 ...     0     0     0]
 [ 5118  2731  3015 ...     0     0     0]
 [ 1269  2392  2530 ...     0     0     0]]
```

```
xtest1=  vect.texts_to_sequences(test['preprocessed_plots'])
xtest = pad_sequences(xest1, maxlen=1200, padding='post')
```

Model creation with Embedding layer to compute word embeddings:

jupyter 2018103058_NLP Last Checkpoint: 16 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [15]:

```
model = keras.Sequential()
model.add(layers.Embedding(vocab_size, output_dim=50, input_length=1200))
model.add(layers.LSTM(128, return_sequences=True))
model.add(layers.Dropout(0.5))
model.add(layers.LSTM(64))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(71, activation='sigmoid'))


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1200, 50)          6109750
_________________________________________________________________
lstm (LSTM)                  (None, 1200, 128)         91648
_________________________________________________________________
dropout (Dropout)            (None, 1200, 128)         0
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                49408
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 71)                4615
=================================================================
Total params: 6,255,421
Trainable params: 6,255,421
Non-trainable params: 0
_________________________________________________________________
```

Compiling and Fitting the model:

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics = METRICS)
```

```
In [17]: history = model.fit(xtrain,y_train,
                             epochs = 10,
                             verbose = 1,
                             validation_data=(xtest, y_test),
                             batch_size=16)
```

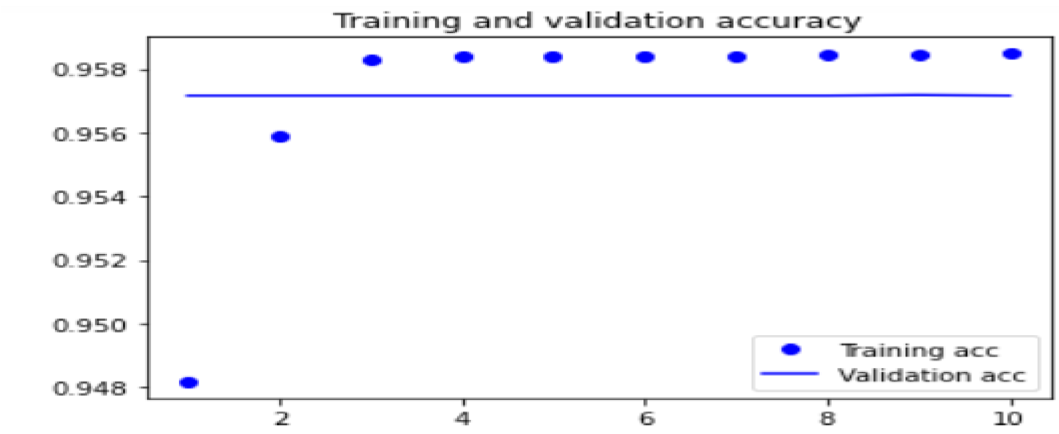
```
Epoch 10/10
594/594 [==============================] - 1593s 3s/step - loss: 0.1413 - tp: 35.4639 - fp: 17.9748 - tn: 324373.2571 - fn: 140
95.8151 - accuracy: 0.9582 - precision: 0.6472 - recall: 0.0023 - auc: 0.8176 - val_loss: 0.1413 - val_tp: 14.0000 - val_fp: 1
4.0000 - val_tn: 201550.0000 - val_fn: 9008.0000 - val_accuracy: 0.9572 - val_precision: 0.5000 - val_recall: 0.0016 - val_auc:
0.8325
```

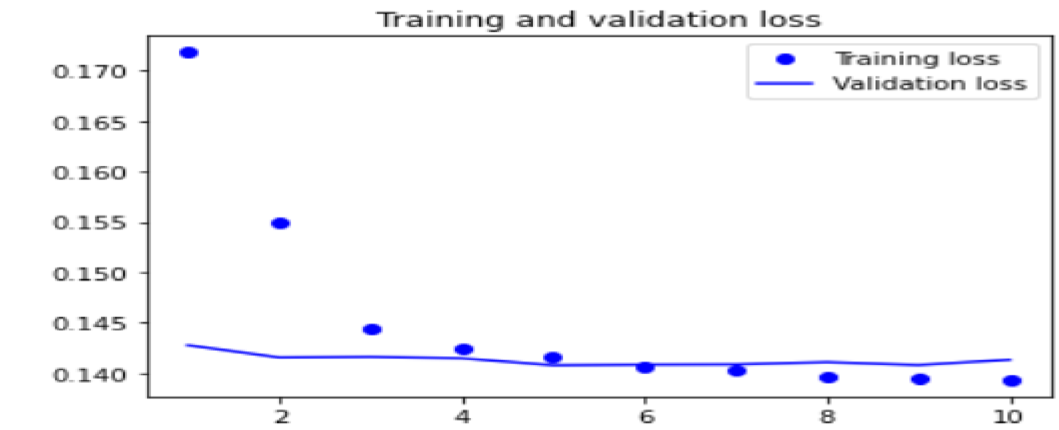
The train accuracy of the model is 95.82% and validation accuracy is 95.72%.

Results:

Train and Validation Graphs:

```
[22]: import matplotlib.pyplot as plt
ac = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(ac) + 1)
plt.plot(epochs, ac, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```






```
Precision: 0.3477, Recall: 0.2321, F1-measure: 0.2784
```

Predict the new data:

In [25]:
```
def predict_sample():
    t = train.sample(1)
    encoded_docs = vect.texts_to_sequences(t['preprocessed_plots'])
    padded_docs = pad_sequences(encoded_docs, maxlen=1200, padding='post')
    pred = model.predict(padded_docs).tolist()
    for i in range(len(pred[0])):
        if(pred[0][i] < 0.1):
            pred[0][i] = 0
        else:
            pred[0][i] = 1

    print("Original tags -->", t['tags'].values)
    print("Predicted tags -->", vectorizer.inverse_transform(pred[0])[0])

predict_sample()
```

```
Original tags --> ['violence']
Predicted tags --> ['comedy' 'cult' 'flashback' 'murder' 'psychedelic' 'revenge' 'romantic'
 'violence']
```

Hyperparameter Tuning:

From the first model one LSTM layer is removed.Number of epochs is 5 and batch size is increased to 64.

In [31]:
```
model = keras.Sequential()
model.add(layers.Embedding(vocab_size, output_dim=50, input_length=1200))
model.add(layers.LSTM(64))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(71, activation='sigmoid'))


model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1200, 50)          6109750
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                29440
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 71)                4615
=================================================================
Total params: 6,143,805
Trainable params: 6,143,805
Non-trainable params: 0
_________________________________________________________________
```

jupyter 2018103058_DL_NLP (unsaved changes)

Logou

File Edit View Insert Cell Kernel Widgets Help

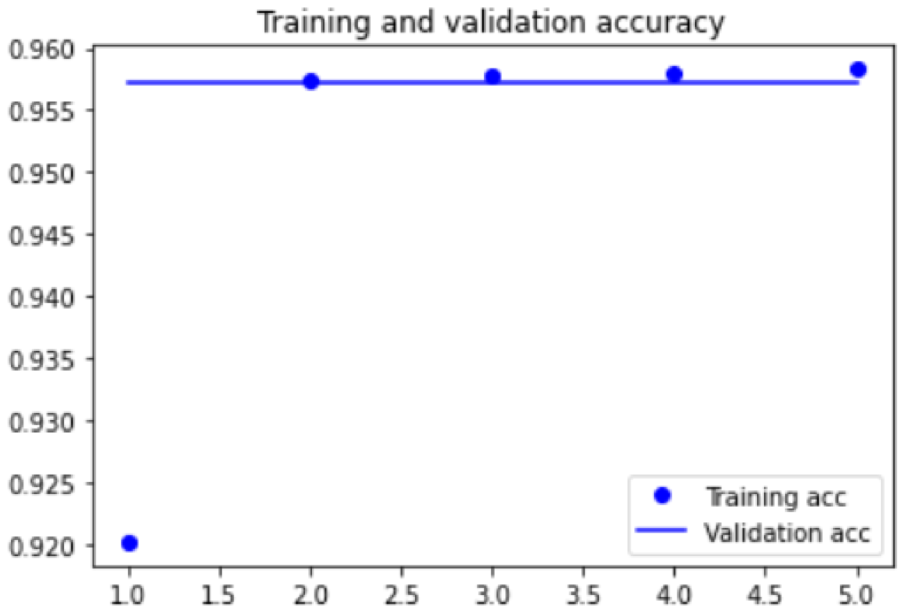
Not Trusted | Python 3

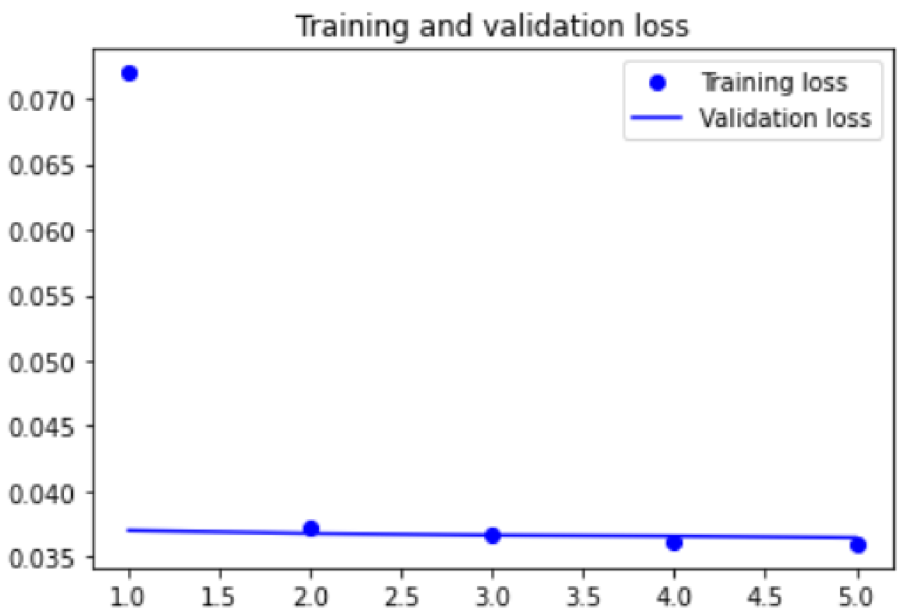
Run Code

In [34]:
```
history = model.fit(padded_docs_train,y_train,
                    epochs = 5,
                    verbose = 1,
                    validation_data=(padded_docs_test, y_test),
                    batch_size=64)
```

```
Epoch 1/5
149/149 [==============================] - 191s 1s/step - loss: 0.1285 - tp: 1467.5667 - fp: 24858.3200 - tn: 303733.0800 - fn: 12938.2467 - accuracy: 0.8451 - precision: 0.0546 - recall: 0.1432 - auc: 0.5665 - val_loss: 0.0370 - val_tp: 0.0000e+00 - val_fp: 0.0000e+00 - val_tn: 201564.0000 - val_fn: 9022.0000 - val_accuracy: 0.9572 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7769
Epoch 2/5
149/149 [==============================] - 258s 2s/step - loss: 0.0375 - tp: 467.0933 - fp: 844.9200 - tn: 327898.8733 - fn: 13786.3267 - accuracy: 0.9574 - precision: 0.3616 - recall: 0.0346 - auc: 0.7345 - val_loss: 0.0368 - val_tp: 0.0000e+00 - val_fp: 0.0000e+00 - val_tn: 201564.0000 - val_fn: 9022.0000 - val_accuracy: 0.9572 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8019
Epoch 3/5
149/149 [==============================] - 259s 2s/step - loss: 0.0370 - tp: 298.8933 - fp: 519.0733 - tn: 328083.2133 - fn: 14096.0333 - accuracy: 0.9576 - precision: 0.3700 - recall: 0.0214 - auc: 0.7564 - val_loss: 0.0367 - val_tp: 0.0000e+00 - val_fp: 0.0000e+00 - val_tn: 201564.0000 - val_fn: 9022.0000 - val_accuracy: 0.9572 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8152
Epoch 4/5
149/149 [==============================] - 257s 2s/step - loss: 0.0364 - tp: 158.5667 - fp: 288.5667 - tn: 328410.8067 - fn: 14139.2733 - accuracy: 0.9579 - precision: 0.3452 - recall: 0.0108 - auc: 0.7674 - val_loss: 0.0366 - val_tp: 0.0000e+00 - val_fp: 0.0000e+00 - val_tn: 201564.0000 - val_fn: 9022.0000 - val_accuracy: 0.9572 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8202
Epoch 5/5
149/149 [==============================] - 261s 2s/step - loss: 0.0360 - tp: 58.7933 - fp: 84.6733 - tn: 328656.1933 - fn: 14197.5533 - accuracy: 0.9585 - precision: 0.4263 - recall: 0.0037 - auc: 0.7772 - val_loss: 0.0365 - val_tp: 0.0000e+00 - val_fp: 0.0000e+00 - val_tn: 201564.0000 - val_fn: 9022.0000 - val_accuracy: 0.9572 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8232
```

The train accuracy is 95.85% and validation accuracy is 95.72%.





There is no large difference between both the models.

Conclusion:

Thus movies are classified into genres based on the plot.

References:

- [Movie Genre Prediction Using Multi Label Classification (analyticsvidhya.com)]
- [Simple Text Classification using Keras Deep Learning Python Library - Step By Step Guide | opencodez]

- Dataset:https://www.kaggle.com/cryptexcode/mpst-movie-plot-synopses-with-tags
- LSTM: Understanding the Number of Parameters | by Murat Karakaya | Deep Learning Tutorials with Keras | Medium