A sample Implementation of LSTM+CNN for Extremist Classification

A sample implementation of the proposed architecture that is LSTM+CNN is performed using Anaconda which provides free Jupyter notebook environment and also gives assistance about python 2.7 and 3.6 . The Jupyter notebook contains a series of cell that carry workable code, output, and descriptive text. For example, the code snapshot in Fig. S1 presents an executable code cell and its output.

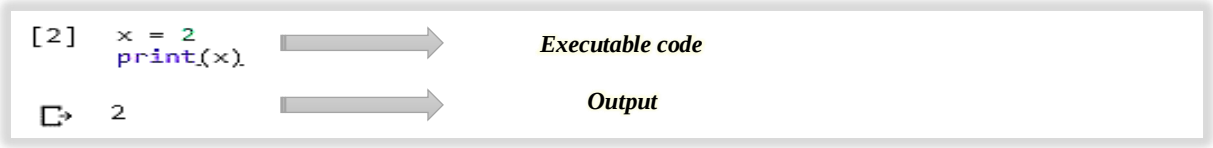

Fig. S1 A Code Cell with an Output

The description about the code executed in the implementation section will be given as follows:

Import certain Libraries: In Fig. S2 certain libraries are imported that are needed to perform different functions and tasks.

```
import pandas as pd
import numpy as np
import re
import collections
import matplotlib.pyplot as plt
# Packages for data preparation
from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
from keras.preprocessing.text import Tokenizer
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import LabelEncoder
# Packages for modeling
from keras import models
from keras import layers
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from keras.layers.convolutional import Conv1D,MaxPooling1D
from keras.preprocessing import sequence
from keras.layers import Dense, Flatten, Activation, Dropout
from keras import regularizers
```

Fig. S2 Import the required libraries

Pandas Data Frame for Data Loading: After importing the pandas data frame the data is loaded into pandas data frame as shown in Fig. S3.

```
df = pd.read_csv("OS(1000).csv")
df.head()
```

Fig. S3 Data Loading into Pandas Data Frames

Whereas the dataset required for the experimentation is presented in Fig. S4 which is the output of the above code snippets (Fig. S3)

	Tweet	label
0	Ottawa police are confirming a shooting at the...	Ex
1	Passersby working on shooting victim. From the...	Ex
2	Shooting at the war memorial in Ottawa. http:\...	Ex
3	After today we will officially be done shootin...	Ex
4	KEY: Nothing in Wilson's story in new autopsy ...	N-Ex

Fig. S4 Required Dataset

Data-preprocessing: -

In order to obtain better feature from our data we will do some of the basic pre-processing steps. So let's discuss them one by one.

Case Conversion: Fig. S5 depicts that case of all the tweets/reviews in the dataset are transformed into lowercase and it is the first step of preprocessing.

```
df['Tweet'] = df['Tweet'].apply(lambda x: " ".join(x.lower() for x in x.split()))
df['Tweet'].head()
```

```
0    ottawa police are confirming a shooting at the...
1    passersby working on shooting victim. from the...
2    shooting at the war memorial in ottawa. http:\... (http:\...)
3    after today we will officially be done shootin...
4    key: nothing in wilson's story in new autopsy ...
Name: Tweet, dtype: object
```

Fig. S5 Case Conversion

Removing Punctuation: Next punctuation is removed from text data to reduce the size of the data which will help to reduce extra information as shown in Fig. S6.

```
df['Tweet'] = df['Tweet'].str.replace('[^\w\s]','')
df['Tweet'].head()
```

```
0    ottawa police are confirming a shooting at the...
1    passersby working on shooting victim from the ...
2    shooting at the war memorial in ottawa httptco...
3    after today we will officially be done shootin...
4    key nothing in wilsons story in new autopsy sa...
Name: Tweet, dtype: object
```

Fig. S6 Remove punctuation

Removal of Stop Word: We used a predefined library as shown in Fig. S7 to remove stopwords from our data as these commonly occurring word will affect the performance of the model.

```
from nltk.corpus import stopwords
stop = stopwords.words('english')
df['Tweet'] = df['Tweet'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
df['Tweet'].head()
```

```
0    ottawa police confirming shooting war memorial...
1    passersby working shooting victim cbcs jason h...
2       shooting war memorial ottawa httptco1z0p4mj5vw
3    today officially done shooting surrendermusicv...
4    key nothing wilsons story new autopsy says fea...
Name: Tweet, dtype: object
```

Fig. S7 Stopword removal.

Rare words removal: Just as we remove stop words we have remove the rare words to improve the model performance depicted in Fig. S8.

```
freq = pd.Series(' '.join(df['Tweet']).split()).value_counts()[-10:]
freq
```

```
ottawau201d          1
commentary           1
oldest               1
canberra             1
reasons              1
httptcojcxzycy9lo    1
hour                 1
timing               1
httptcof1yzhma6cq    1
la                   1
dtype: int64
```

Fig. S8 Rare word removal.

Splitting Data into Train and Test: After data preprocessing, the dataset is arranged into train and test set using scikit learn "*train_test_split"* [21] method. The purpose of train set is to train the model so that it will be able to learn the data and the test set is used to evaluate the performance of the model. Fig. S9 illustrates the code for train test split of data.

```
X_train, X_test, y_train, y_test = train_test_split(df.Tweet, df.label, test_size=0.2, random_state=39)
print('# Train data samples:', X_train.shape[0])
print('# Test data samples:', X_test.shape[0])
assert X_train.shape[0] == y_train.shape[0]
assert X_test.shape[0] == y_test.shape[0]
```

```
# Train data samples: 801
# Test data samples: 201
```

Fig. S9 Train Test Split of Data

Tokenization: Tokenization is the process in which sentences divided into words called as tokens. The tokenization is performed using keras Tokenizer API (Fig. S10).

```
# Preprocess text with Keras for extremist classification
from keras.preprocessing.text import Tokenizer
max_features = 10000 ## Parameter indicating the number of words we'll put in the dictionary
max_len = 40
# Define Tokenizer
tk = Tokenizer(num_words=max_features)
# Fit Tokenizer on text (Build vocab etc..)
tk.fit_on_texts(X_train)
word_index = tk.word_index
print('Found {:,} unique words.'.format(len(word_index)))
print('Fitted tokenizer on {} documents'.format(tk.document_count))
print('{} words in dictionary'.format(tk.num_words))
print('Top 5 most common words are:', collections.Counter(tk.word_counts).most_common(5))
```

The output of above code snippets is as follows:

```
Found 2,184 unique words.
Fitted tokenizer on 801 documents
10000 words in dictionary
Top 5 most common words are: [('ottawa', 446), ('parliament', 276), ('shooting', 191), ('hill', 166), ('shot', 152)]
```

Fig. S10 Dataset Tokenization.

After having created the vocabulary we can convert the text to a list of integer indexes. This is done with the text_to_sequences method of the Tokenizer. Fig. S11 depicts text to sequence of integers conversion.

```
X_train_seq = tk.texts_to_sequences(X_train)
X_test_seq = tk.texts_to_sequences(X_test)

print('"{}" is converted into {}'.format(X_train[0], X_train_seq[0]))
```

```
"ottawa police confirming shooting war memorial minutes ago info cbcott ottnews" is converted into [204, 64, 917, 520, 918, 919, 920, 921, 3, 922, 9
23, 924, 925, 926, 284, 927]
```

Fig. S11 Text to sequence of integers

Converting the target classes to numbers: We need to convert the target classes to numbers as well as shown in Fig S12.

```
le = LabelEncoder()
y_train_le = le.fit_transform(y_train)
y_test_le = le.transform(y_test)
```

Fig. S12 Conversion of target classes

Building a LSTM+CNN Learning Model: Fig. S13 depicts the code snapshots regarding the development of LSTM+CNN model. In order to build LSTM-CNN model some important layers are also imported. The first layer is the embedding layer which acts as an input layer for the model. It consists of three parameters which are:

- ***max_features(input_dim):*** max_features also known as input dimension represents the vocabulary size. It is the amount of top words selected from the dataset.
- ***embed_dim(output_dim):*** It is also known as output dimension which describe the length of embedding vector. In our case, the value of embed_dim is 128 which means the size of vector is 128.

- ***input_length:*** It shows the size of individual input sequences /post.

The next layer is the LSTM layer which is the first hidden layer in LSTM-CNN model. This layer contain a parameter.

- ***LSTM units:*** Size of LSTM hidden state.

Then there is a convolutional layer which is the second hidden layer in CNN model. This layer contains following parameters.

- ***filters:*** It is the amount of output filter within the convolutional layer.
- ***kernel_size:*** It determine 1D convolutional window length.
- ***padding:*** It has different values like *"valid"*, *"casual"* or *"same"*. When padding is same then the length of original input and output is same and when the value of padding is casual it produces widened convolutions. In case of valid it means no padding.
- ***activation:*** relu activation function is used and it is a nonlinear operation.

The third layer is the maxpooling layer. It contain the argument which are stated as:

- ***pool_size:*** It specify maxpooling window size.

Flatten is the next layer after maxpooling layer and its purpose is to create the column/single vector of a pooled feature map. After flatten the dense layer is used, which contain the sigmoid activation function. The mathematical equation of sigmoid function is

$\sigma(x) = 1/1 + e^{-x}$

In sigmoid function the range of output probabilities will be from 0 to 1. Moreover, the compile method build the model for training. It involves the following parameters:

- ***loss:*** It is an optimization score function.
- ***optimizer:*** It presents an instance of an optimizer.
- ***metrics:*** This parameter contains the metrics used for an evaluation.

Lastly, the summary method will depict the summary of the model.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM,Bidirectional
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import Dropout
from keras import optimizers
import keras
from keras.utils import plot_model
# create the model
embed_dim = 128
model = Sequential()
model.add(Embedding(max_features, embed_dim,input_length = max_len))
model.add(Dropout(0.5))
model.add(LSTM(80,return_sequences = True))
model.add(Conv1D(filters=32, kernel_size=2, padding='valid', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adamax', metrics=['accuracy'])
print(model.summary())
```

Fig. S13 Developing a Deep Learning Model

Model Summary: Fig. S14 shows the summary of the model which is generated using "*print.summary()"* function.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_70 (Embedding)     (None, 40, 128)           1280000
_________________________________________________________________
dropout_58 (Dropout)         (None, 40, 128)           0
_________________________________________________________________
lstm_60 (LSTM)               (None, 40, 80)            66880
_________________________________________________________________
conv1d_35 (Conv1D)           (None, 39, 32)            5152
_________________________________________________________________
max_pooling1d_35 (MaxPooling (None, 19, 32)            0
_________________________________________________________________
flatten_35 (Flatten)         (None, 608)               0
_________________________________________________________________
dense_68 (Dense)             (None, 1)                 609
=================================================================
Total params: 1,352,641
Trainable params: 1,352,641
```

Fig. S14 Summary of Model

Fitting the Model: The "*model.fit()"* function is used to train the model on the training data as shown in Fig. S15. The parameters of the "*model.fit()"* method is given as follows:

- ***X_train_seq:*** It covers training data.
- ***Y_train_le:*** It covers a label (target) data.
- ***epochs:*** This parameter contain epochs number for model training. It is the amount of iteration over the complete training samples.
- ***Validation_split:*** The validation data is the parameter of the "*model.fit()"* that contains the data which will not pass through the training procedure.
- ***Batch Size*:** The batch size is a hyperparameter which is an amount of training samples per forward or backward pass. Its default value is 32 (in Fig. S15).

```
batch_size = 32
history=model.fit(X_train_seq,y_train_le, epochs =4, batch_size=batch_size,validation_split=0.1)
```

Fig. S15 Model Fitting

Training Output: The output of the model during training is illustrated in Fig. S16. It shows the accuracy, validation accuracy and the loss associated with both the metrics. Moreover, the model is trained on 720 samples and validate on 81 samples.

```
Train on 720 samples, validate on 81 samples
Epoch 1/4
720/720 [==============================] - 10s 13ms/step - loss: 0.6889 - acc: 0.6014 - val_loss: 0.6700 - val_acc: 0.7531
Epoch 2/4
720/720 [==============================] - 2s 2ms/step - loss: 0.6100 - acc: 0.8111 - val_loss: 0.4832 - val_acc: 0.8148
Epoch 3/4
720/720 [==============================] - 2s 2ms/step - loss: 0.3693 - acc: 0.8639 - val_loss: 0.3524 - val_acc: 0.8395
Epoch 4/4
720/720 [==============================] - 2s 2ms/step - loss: 0.2876 - acc: 0.8861 - val_loss: 0.3185 - val_acc: 0.8642
```

Fig. S16 Output during Training

Model Evaluation: In Fig. S17, the performance of the model on a test set is evaluated using accuracy evaluation metric. The method used to perform evaluation is "*model.evaluate()"* whereas verbose is an argument of the evaluate function which contain an integer values like 0,1,2 where 0 means silent,1 means progress bar,2 means 1 line per epoch.

```
score,acc = model.evaluate(X_test_seq, y_test_le, verbose = 2, batch_size = batch_size)
print("score: %.2f" % (score))
print("acc: %.2f%%" % (acc*100))
```

```
score: 0.34
acc: 87.06%
```

Fig. S17 Evaluation of the model

Confusion Matrix: Confusion matrix is represented in the form of a table which is used to report the performance related with the model [23]. In order to use the confusion matrix it is necessary to import it from sci-kit learn library (Fig. S18).

```
%matplotlib inline
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt
```

Fig. S18 Import Confusion Matrix

The code snapshot in Fig. S19 shows that the creation of a variable cm is performed through calling a "*confusion_matrix*". The "*confusion_matrix"* contains two parameters y_test_le and y_pred. So, a confusion matrix named as cm is defined in the below screenshot.

```
cm=confusion_matrix((y_test_le), y_pred)
```

Fig. S19 Creation of Confusion Matrix

A confusion matrix is plotted using "*plot_confusion_matrix"* function (Fig. S20).

```
class_names=["NR","R"]
print(__doc__)

import itertools
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.rainbow):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test_le, y_pred)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,title='Confusion matrix')

# Plot normalized confusion matrix
#plt.figure()
#plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,title='Normalized confusion matrix')

plt.show()
```

Fig. S20 Code on Plotting Confusion Matrix

Now, the output of the code shown in Fig. 20 is illustrated in Fig. S21. In the plot (Fig. S21), predicted label is placed at x-axis and True label is placed at y-axis. The red cells depicts the accurate predictions and blue cells depicts inaccurate predictions.

```
Automatically created module for IPython interactive environment
Confusion matrix, without normalization
[[89 11]
 [15 86]]
```

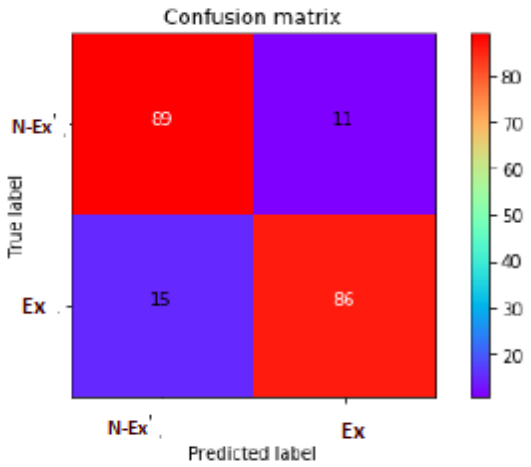

Fig. S21 Confusion Matrix

Evaluation Metrics: Fig. S22 presents the code and its desired output for an evaluation metrics namely precision, recall and f1-score. The function "*model.predict()"* is used to perform prediction on test set.

	precision	recall	f1-score	support
N-Ex	0.86	0.89	0.87	100
Ex	0.89	0.85	0.87	101
avg / total	0.87	0.87	0.87	201

Fig. S22 Metrics for Evaluation.

==========

Muhammad Zubair Asghar, PhD

ORCID: https://orcid.org/0000-0003-3320-2074

Google Scholar: https://scholar.google.com.pk/citations?user=_CNMYU0AAAAJ&hl=en

HEC Approved PhD Supervisor, ICIT, Gomal University, khyber Pakhtunkhwa (KP), Pakistan