National College of Ireland

Configuration Manual

Named Entity Recognition on Kannada Low Resource Languages Using Deep Learning Models

MSc in Data Analytics

Pavan Kulkarni

Student ID: x19231075

School of Computing

National College of Ireland

Supervisor:

Prof Dr. Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Pavan Kulkarni

Student ID: X19231075

Programme: MSc. Data Analytics **Year:** 2020-2021

Module: Research Project

Lecturer: Dr. Christian Horn
Submission Due Date: 16/08/2020

Project Title: Named Entity Recognition on Kannada Low Resource using Deep Learning and Machine Learning Languages

Word Count: **1568** **Page Count: 22**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Pavan Kulkarni

Date: 16/08/2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Pavan Kulkarni
Student ID: x19231075

1 Introduction

This documentation takes us to necessary steps and preparation required to implement and run this proposed model. Here, the systems requirements used, and hardware configuration of the system is explained. The minimum system specification requirement necessary to run this model is explained.

2 System Prerequisites

2.1 Configuration of Hardware used

The setup of the Lenovo laptop that was utilized for the investigation is shown in Figure 1. The laptop is equipped with an Intel Core i3-8145U CPU, 12 GB of Random-Access Memory (RAM) and 1 Terra Bytes hard disk. The operating system (OS) is Windows 10 Home edition.

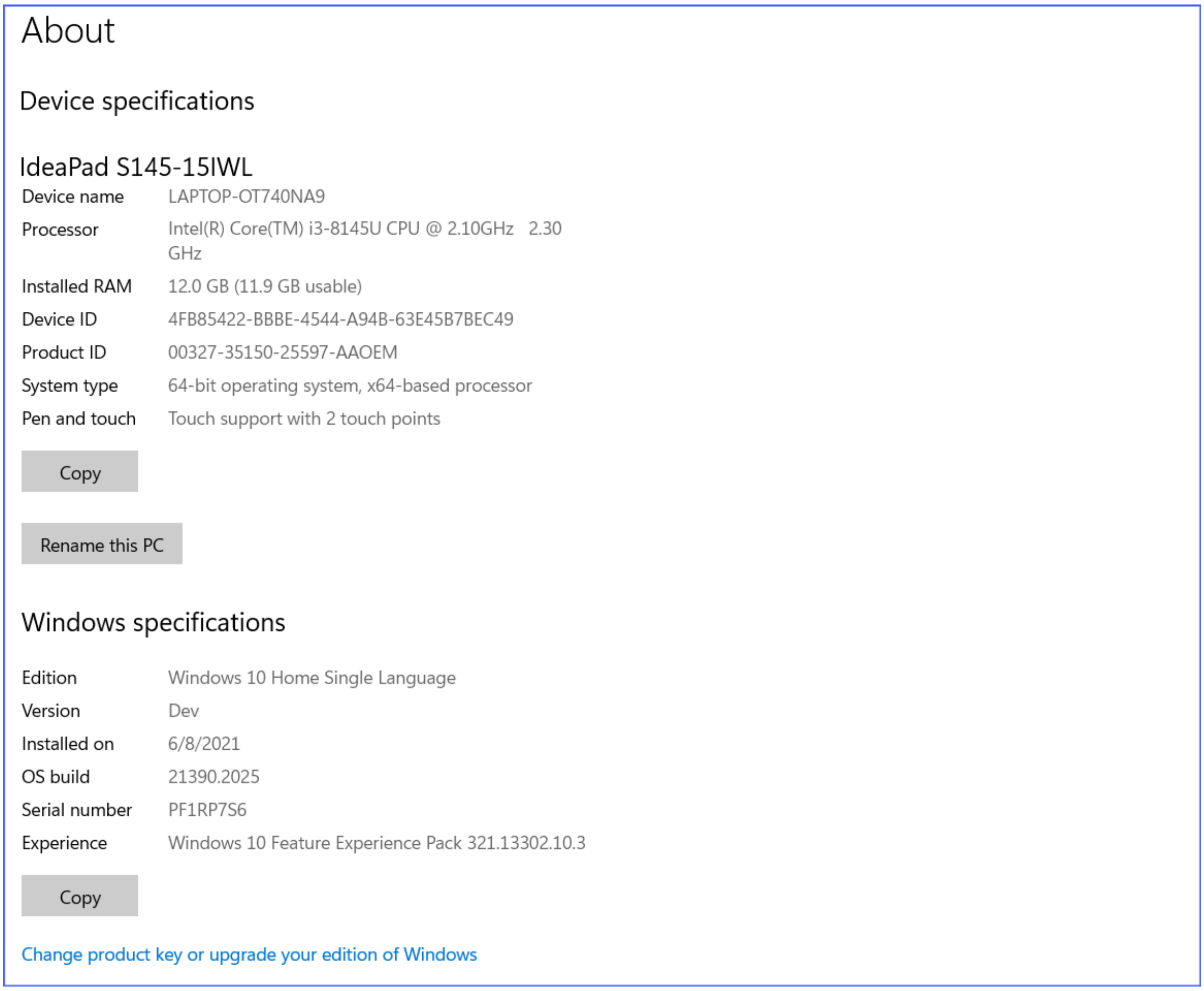

Figure 1. The configuration of system used to run the model.

2.2 Configuration of Software

Google Collaboratory and Jupyter Notebook (Anaconda 3) tools are used to carry out the project. The following section summarizes all of the procedures involved in downloading and installing the software packages requirement.

3 Setup of the Environment

3.1 Google Notebook for Collaboration

Some of the packages to run BERT model found to be not compatible in the Jupyter Notebook (Anaconda 3). So, Google Collaboratory is used to run Google's BERT deep learning model. And Bi-LSTM, Random Forest Classifier have been run successfully on Jupyter Notebook (Anaconda 3) without any issues. The following steps will explains how to create the preliminary setup to run these models.

1. Gmail account is necessary to run model on Google Collaboratory.
2. Click this link after signup and follow the instructions.

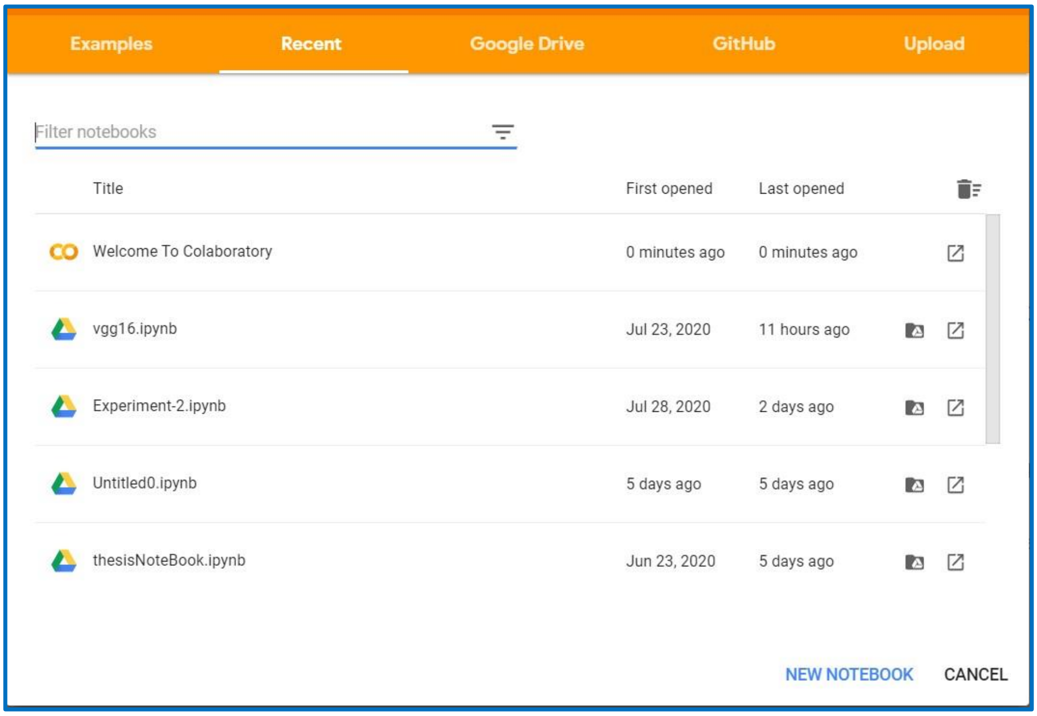

Figure 2: Create a new notebook

3. To utilize the existing notebooks, select New Notebook or Upload.
4. Change the notebook type to GPU after establishing a connection. This is done by clicking on Runtime and then on Modify Runtime.

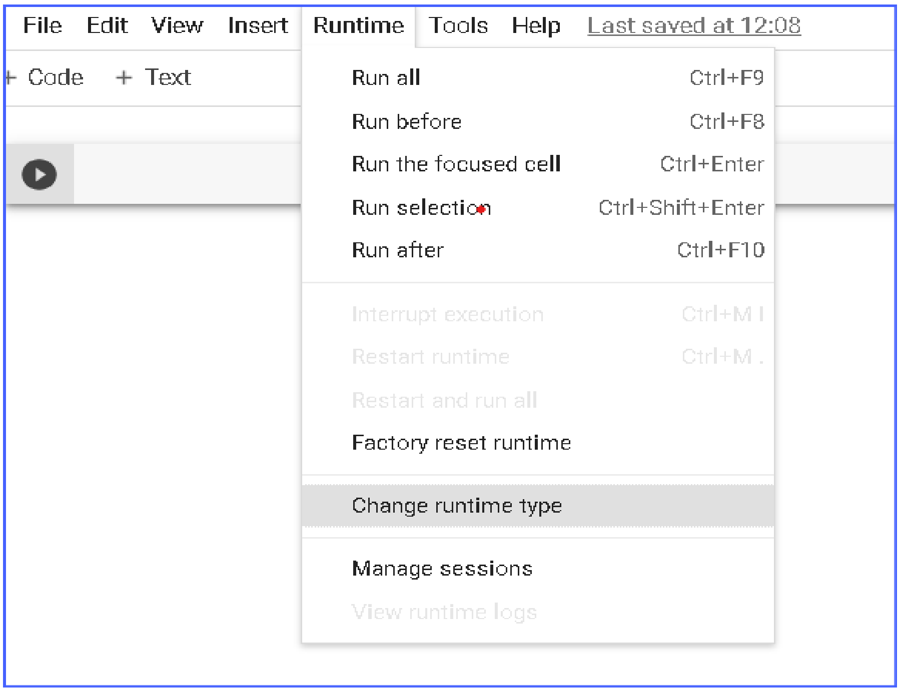

Figure 3. Based on the requirements change the runtime type.

5. Then click Save after selecting GPU.

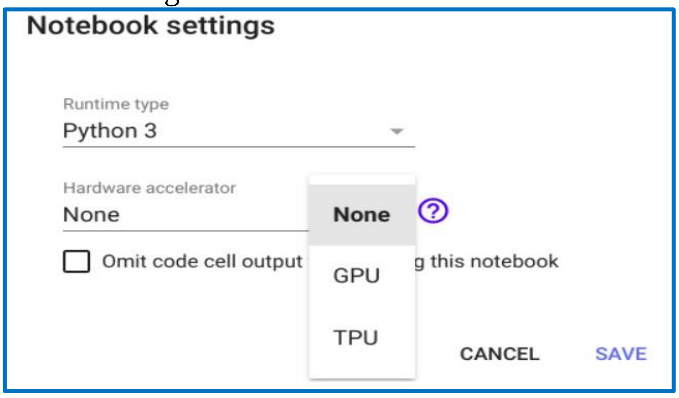

Figure 4: Settings for the Runtime

6. Then select Connect to hosted runtime from the dropdown menu of the Connect button.

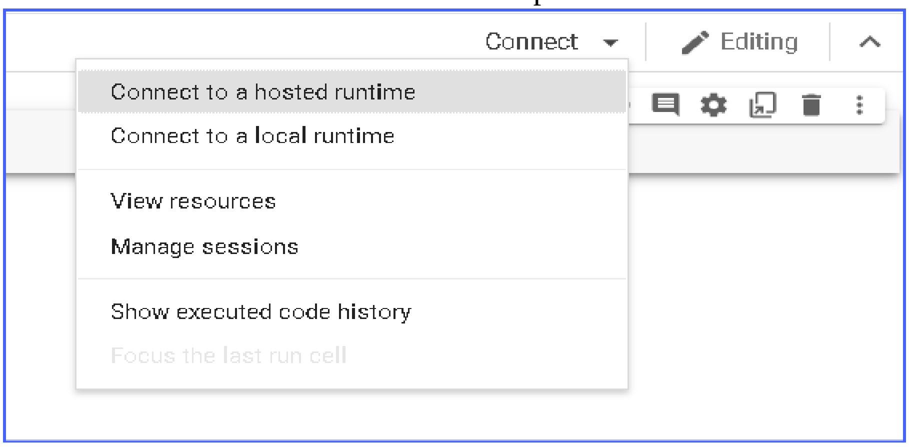

Figure 5. Use Google's Infrastructure to connect to it.

3.2 Jupyter Notebook (Anaconda 3)

To install Jupyter Notebook (Anaconda 3), go to this page and click the "download" button to begin the process. All the download and installation processes are shown in Figure 6.

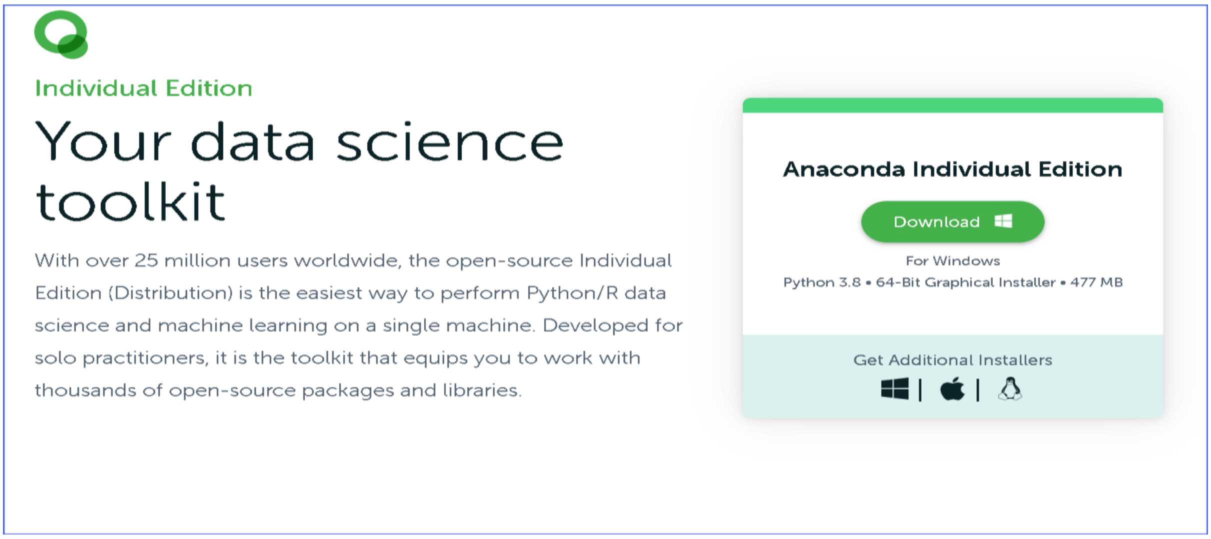

Figure 6. Anaconda download for windows edition.

After the installation, from start, select Jupyter Notebook (Anaconda 3) from the Anaconda 3 64 bit drop down icon.

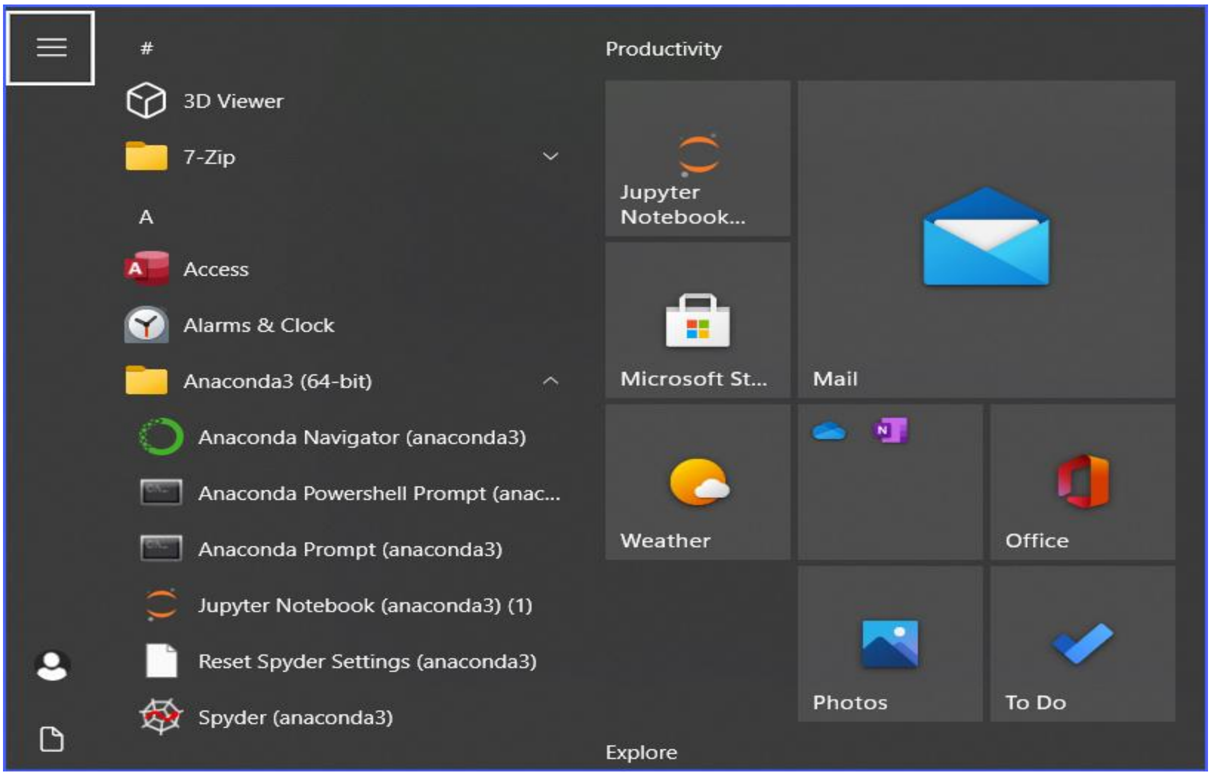

Figure 7. From windows start function select the Jupyter Notebook (anaconda 3) (1)

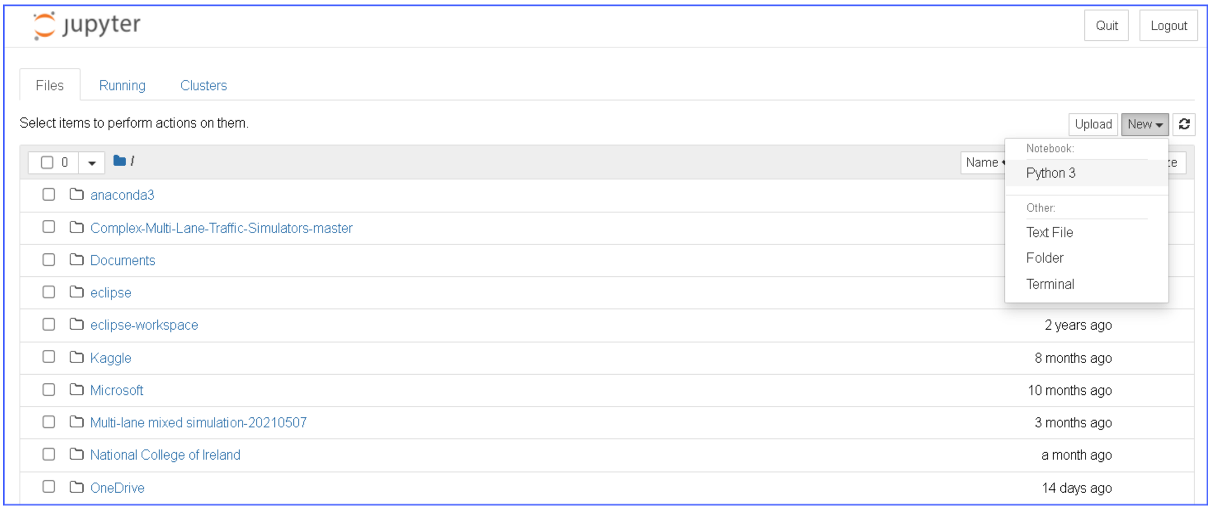

Figure 8. Create a new blank python 3 notebook from drop down button.

4 Implementation

4.1 Jupyter Notebook data source

4.1.1Random Forest Classifier model:

The dataset is separated by space and file is encoded with ‘utf-8’ format.

```
#read the input file name "input.txt", separated by space and unicode encoding 'utf-8'
df = pd.read_csv(os.path.join('input.txt'),delimiter="\t", encoding = "utf-8")
index = [i for i in range(0,len(df['Word']))]
#populating the missing values with default na values
df = df.fillna(method='ffill')
```

Figure 9. Reading the file using pandas and populating missing values.

```
df.columns = df.iloc[0]
df = df[1:]
#selecting first 5 columns from the input file
df.columns = ['0','Sentence#','Word','POS','Tag']
df = df.reset_index(drop=True)
#printing first five rows of the input file
df.head()
```

Figure 10. Select the required columns.

```
#printing datatype of the dependent and independent variables in the input file
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2070 entries, 0 to 2069
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   0          2070 non-null   int64
 1   Sentence#  2070 non-null   float64
 2   Word       2070 non-null   object
 3   POS        2070 non-null   object
 4   Tag        2070 non-null   object
dtypes: float64(1), int64(1), object(3)
memory usage: 81.0+ KB
```

Figure 11. The datatype of the columns used.

```
#class getsentence takes dependent variables like 'words', 'pos','tag' and 'sentence#'
#based on sentence no. the data will be grouped and read the full data based on lambda function
class getsentence(object):

    def __init__(self, data):
        self.n_sent = 1.0
        self.data = data
        self.empty = False
        agg_func = lambda s: [(w, p, t) for w, p, t in zip(s["Word"].values.tolist(),
                                                           s["POS"].values.tolist(),
                                                           s["Tag"].values.tolist())]
        self.grouped = self.data.groupby("Sentence#").apply(agg_func)
        self.sentences = [s for s in self.grouped]
```

Figure 12. This is the main function 'sentence getter'. Group the sentence based on the lambda function.

```
#select only the sentence from the getter function, where we are passing the whole data
sentences = getter.sentences
#ths is how first sentence will look like.
print(sentences[1])
```

```
[('ಇದು', 'PRP', 'O'), ('೧೮೫೭ರಿಂದ', 'NN', 'O'), ('೧೯೪೭ರ', 'NN', 'B-DATE'), ('ಆಗಸ್ಟ್', 'NN', 'B-DATE'), ('೧೫ರವರೆಗೆ',
'NN', 'O'), ('ನಡೆದ', 'NN', 'O'), ('ಭಾರತದ', 'NNP', 'B-LOC'), ('ವಿವಿಧ', 'JJ', 'O'), ('ಸಂಘ-ಸಂಸ್ಥೆಗಳ', 'NN', 'O'), ('ತ
ತ್ವಗಳು,', 'NN', 'O'), ('ದಂಗೆಗಳು,', 'NN', 'O'), ('ಹೋರಾಟಗಳು', 'NN', 'O'), ('ಮತ್ತು', 'CC', 'O'), ('ಪ್ರಾಣಾಹುತಿಗಳ', 'NN',
'O'), ('ಸಂಗಮ', 'NN', 'O'), ('.', 'SYM', 'O')]
```

Figure 13. The first sentence of the data with 'word', 'POS' and 'Tag' entities.

```
#printing only the 'tag' with location mentioned words and displaying first five records
data.loc[data['Tag'] == 'B-LOC', 'Word'].head()
```

```
16      ಭಾರತದ
30     ಪ್ಲಾಸಿ
62      ಭಾರತದ
68     ಪ್ಲಾಸಿ
71       ನೂರು
Name: Word, dtype: object
```

Figure 14. The words tagged with 'B-LOC' is separated from the dataset.

```
#printing only the 'tag' with organization names
data.loc[data['Tag'] == 'I-ORG', 'Word'].head()
```

```
34              ಇಂಡಿಯ
35             ಕಂಪನಿಯ
110          ರಾಷ್ಟ್ರೀಯ
111    ಕಾಂಗ್ರೆಸ್ಸೊದಲು
163          ರಾಷ್ಟ್ರೀಯ
Name: Word, dtype: object
```

Figure 15. Function extracted the words with 'I-ORG' from the input file.

```
#printing only the 'tag' with person names
data.loc[data['Tag'] == 'I-PER', 'Word'].head()
```

```
155     ಗಾಂಧಿಯವರ
199       ಬೋಸ್ರು
278      ಸಿಯಾರನು
424     ಟಿಪ್ಪುವಿನ
605          ರಾವ್
Name: Word, dtype: object
```

Figure 16. Person names are tagged separately from the dataset.

```
#printing word with 'O' labels
data.loc[data['Tag'] == 'O', 'Word'].head()
```

```
0          ಸ್ವಾತ್ರಂತ್ರ್ಯ
1               ಚಳುವಳಿಯು
3           ಸಾಮ್ರಾಜ್ಯದಿಂದ
4     ಸ್ವಾತ್ರಂತ್ರ್ಯವನ್ನು
5                 ಪಡೆಯಲು
Name: Word, dtype: object
```

Figure 17. From the dataset 'o' is mentioned for words which are not an entity.

```
#defining the array of the data
def feature_map(word):
    return np.array([word.istitle(), word.islower(), word.isupper(), len(word),
                     word.isdigit(),  word.isalpha()])
```

Figure 18. Converting words into array representation.

```
#printing array of data of first five records
print(words[:5])
```

```
[array([ 0,  0,  0, 13,  0,  0]), array([0, 0, 0, 8, 0, 0]), array([0, 0, 0, 8, 0, 0]), array([ 0,  0,  0, 13,  0,
0]), array([ 0,  0,  0, 18,  0,  0])]
```

Figure 19. Array representation of first five words.

```
#using random forest classifier and predict the classification of words and tags using cross_val_predict
pred = cross_val_predict(RandomForestClassifier(n_estimators=20),X=words, y=tags, cv=5)
```

```
C:\Users\vidya\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:668: UserWarning: The least populated cla
ss in y has only 2 members, which is less than n_splits=5.
  % (min_groups, self.n_splits)), UserWarning)
```

Figure 20. Using Random Forest Classifier assign values to the model.

```
#using classification report package get the result of the test with precision, recall, f1-measure and
#printing the accuracy of the model
from sklearn.metrics import classification_report
report = classification_report(y_pred=pred, y_true=tags)
print(report)
```

```
              precision    recall  f1-score   support

       B-COM       0.00      0.00      0.00        59
      B-DATE       0.78      0.22      0.34        32
       B-LOC       0.00      0.00      0.00       106
       B-NUM       0.00      0.00      0.00        22
       B-ORG       0.00      0.00      0.00        21
       B-PER       0.00      0.00      0.00        71
       B-ROL       0.00      0.00      0.00        30
       I-COM       0.00      0.00      0.00         2
       I-LOC       0.00      0.00      0.00         9
       I-ORG       0.00      0.00      0.00        25
       I-PER       0.00      0.00      0.00        44
       I-ROL       0.00      0.00      0.00         3
           O       0.80      1.00      0.89      1646

    accuracy                           0.80      2070
   macro avg       0.12      0.09      0.09      2070
weighted avg       0.65      0.80      0.71      2070
```

Figure 21. Based on classification report package found the precision, recall, f1-score and model accuracy.

```
#from accuracy score calcualate the accuracy of the model separately.
from sklearn.metrics import accuracy_score
#printing the accuracy of the model using tags and prediction of the model
print(accuracy_score(tags,pred))

0.7985507246376812
```

Figure 22. Accuracy output from the model.

4.1.2 Bi-LSTM Model:

The steps followed to implement the Bi-LSTM model are discussed as follows:

```
#import tensorflow
import tensorflow                    as tf
#from tensorflow inport the modules like model and input
from tensorflow.keras                import Model,Input
#importing tLSTM model, embedding which are essential for testing
from tensorflow.keras.layers         import LSTM,Embedding,Dense
#import bidirectional, timedistributed packages to perform bi-LSTM
from tensorflow.keras.layers         import TimeDistributed, SpatialDropout1D,Bidirectional
#importing the tensorflow pad sequence package
from tensorflow.keras.preprocessing.sequence import pad_sequences
#import categorical package from tensorflow
from tensorflow.keras.utils          import to_categorical
#import train and test dataset split ti train the model
from sklearn.model_selection         import train_test_split
#to perform testing import earlystoping and model checkpoint from keras
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

Figure 23. The important libraries required to import into Jupyter Notebook.

Input file will be read similar to above mentioned model with encoding of 'utf-8'.

```
#set maximum lenth of sentences is 50
max_len = 50
#initialte x calculate total words in the sentence
X = [[word_idx[w[0]] for w in s] for s in sentence]
#using pad sequence function get the words using post padding until last but one word
X = pad_sequences(maxlen = max_len,sequences = X,padding = 'post',value = num_words-1)
#calculate and get tags in the sentence
y = [[tag_idx[w[2]] for w in s] for s in sentence]
#using post padding get total number tags from the sentence
y = pad_sequences(maxlen = max_len,sequences = y,padding = 'post',value = tag_idx['O'])
#categoriese total number of word tags
y = [to_categorical(i,num_classes = num_words_tag) for i in  y]
```

Figure 24. Initializing maximum length of sentences, pad function to get the words until last word and categorize total number of word tags.

```
#train and test the model amd split the test and train model
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.1,random_state=1)
```

Figure 25. Test and train data is split into 10:90 and assign values to test and train variables.

```
#from input word get the shape and maximum lenght of the sentence
input_word = Input(shape = (max_len,))
#perform embeddings on input dimension, output dimension, input lenth
model = Embedding(input_dim = num_words,output_dim = max_len,input_length = max_len)(input_word)
#dropout set to 0.1
model = SpatialDropout1D(0.1)(model)
#call Bi-LSTM model and perform NER on model
model = Bidirectional(LSTM(units=100,return_sequences = True, recurrent_dropout = 0.1))(model)
#using timedistributed and softmax activation perform the dense operation
out = TimeDistributed(Dense(num_words_tag,activation = 'softmax'))(model)
#from inpit and output calculate the test parameters
model = Model(input_word,out)
#print the summary of the model
model.summary()
```

```
Model: "model"

Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 50)]              0

embedding (Embedding)        (None, 50, 50)            64600

spatial_dropout1d (SpatialDr (None, 50, 50)            0

bidirectional (Bidirectional (None, 50, 200)           120800

time_distributed (TimeDistri (None, 50, 13)            2613
=================================================================
Total params: 188,013
Trainable params: 188,013
Non-trainable params: 0
```

Figure 26. The hyperparameters are used in the Bi-LSTM model are mentioned.

```
#using adam optimizer and categorical crossentropy measure the accuracy of the model
#Adam optimization is a stochastic gradient descent method based on adaptive first- and second-order moment estimation.
#When there are two or more label classes, use this crossentropy loss function.
model.compile(optimizer = 'adam',loss = 'categorical_crossentropy',metrics = ['accuracy'])
```

Figure 27. The 'adam' optimizer, 'accuracy' metrics are used during the evaluation.

```
# from livelossplot import PlotLossesKeras
#calculate the val_accuracy of the test
early_stopping = EarlyStopping(monitor = 'val_accuracy',patience =2,verbose = 0,mode = 'max',restore_best_weights = Fals
callbacks = [early_stopping]
#predefining the parameters such as epochs, validation split, verbose parameters
history = model.fit(
    x_train,np.array(y_train),
    validation_split =0.2,
    batch_size = 64,
    epochs = 100,
    verbose =1
)
```

```
Epoch 1/100
2/2 [==============================] - 1s 385ms/step - loss: 2.5523 - accuracy: 0.4179 - val_loss: 2.4830 - val_accu
racy: 0.9497
Epoch 2/100
2/2 [==============================] - 0s 127ms/step - loss: 2.4592 - accuracy: 0.9480 - val_loss: 2.3853 - val_accu
racy: 0.9497
Epoch 3/100
2/2 [==============================] - 0s 131ms/step - loss: 2.3561 - accuracy: 0.9484 - val_loss: 2.2554 - val_accu
racy: 0.9497
Epoch 4/100
2/2 [==============================] - 0s 135ms/step - loss: 2.2114 - accuracy: 0.9484 - val_loss: 2.0480 - val_accu
racy: 0.9497
Epoch 5/100
2/2 [==============================] - 0s 134ms/step - loss: 1.9713 - accuracy: 0.9484 - val_loss: 1.6577 - val_accu
racy: 0.9497
Epoch 6/100
2/2 [==============================] - 0s 137ms/step - loss: 1.4821 - accuracy: 0.9484 - val_loss: 0.8828 - val_accu
racy: 0.9497
Epoch 7/100
2/2 [==============================] - 0s 134ms/step - loss: 0.6838 - accuracy: 0.9484 - val_loss: 0.3331 - val_accu
```

Figure 28. The Bi-LSTM model run with the above-mentioned parameters.

```
#from the above testing evaluate the model and display the accuracy
model.evaluate(x_test,np.array(y_test))

1/1 [==============================] - 0s 2ms/step - loss: 0.2265 - accuracy: 0.9624

[0.22653743624687195, 0.9623529314994812]
```

Figure 29. The output of the Bi-LSTM model with accuracy of 0.9623 and loss of 0.2265.

Experiment 1:

The aim of this activity is to improve the Bi-LSTM model and increase the accuracy of the model. By changing the parameters to find the best fit model. The steps followed are given as follows.

```
#from input word get the shape and maximum lenght of the sentence
input_word = Input(shape = (max_len,))
#perform embeddings on input dimension, output dimension, input lenth
model = Embedding(input_dim = num_words,output_dim = max_len,input_length = max_len)(input_word)
#dropout set to 0.05
model = SpatialDropout1D(0.05)(model)
#call Bi-LSTM model and perform NER on model
model = Bidirectional(LSTM(units=50,return_sequences = True, recurrent_dropout = 0.05))(model)
#using timedistributed and softmax activation perform the dense operation
out = TimeDistributed(Dense(num_words_tag,activation = 'softmax'))(model)
#from inpit and output calculate the test parameters
model = Model(input_word,out)
#print the summary of the model
model.summary()

Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 50)]              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 50, 50)            64600
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 50, 50)            0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 50, 100)           40400
_________________________________________________________________
time_distributed_1 (TimeDist (None, 50, 13)            1313
=================================================================
Total params: 106,313
Trainable params: 106,313
Non-trainable params: 0
_________________________________________________________________
```

Figure 30. The parameters used during this test are given in the above figure.

```
# from livelossplot import PlotLossesKeras
#calculate the val_accuracy of the test
early_stopping = EarlyStopping(monitor = 'val_accuracy',patience =2,verbose = 0,mode = 'max',restore_best_weights = Fal
callbacks = [early_stopping]
#predefining the parameters such as epochs, validation split, verbose parameters
history = model.fit(
    x_train,np.array(y_train),
    validation_split =0.5,
    batch_size = 32,
    epochs = 50,
    verbose =1
)

Epoch 1/50
3/3 [==============================] - 1s 208ms/step - loss: 2.5504 - accuracy: 0.1895 - val_loss: 2.4719 - val_accu
racy: 0.9506
Epoch 2/50
3/3 [==============================] - 0s 48ms/step - loss: 2.4470 - accuracy: 0.9458 - val_loss: 2.3557 - val_accur
acy: 0.9514
Epoch 3/50
3/3 [==============================] - 0s 49ms/step - loss: 2.3254 - accuracy: 0.9458 - val_loss: 2.1993 - val_accur
acy: 0.9514
Epoch 4/50
3/3 [==============================] - 0s 50ms/step - loss: 2.1495 - accuracy: 0.9458 - val_loss: 1.9540 - val_accur
acy: 0.9514
Epoch 5/50
3/3 [==============================] - 0s 56ms/step - loss: 1.8698 - accuracy: 0.9458 - val_loss: 1.5229 - val_accur
acy: 0.9514
Epoch 6/50
3/3 [==============================] - 0s 60ms/step - loss: 1.3684 - accuracy: 0.9458 - val_loss: 0.8274 - val_accur
acy: 0.9514
Epoch 7/50
3/3 [==============================] - 0s 49ms/step - loss: 0.6879 - accuracy: 0.9458 - val_loss: 0.3562 - val_accur
```

Figure 31. The parameters used during tuning the model are epochs of 50, batch size is 32 and other values are given in the screen shot.

```
#from the above testing evaluate the model and display the accuracy
model.evaluate(x_test,np.array(y_test))
```

```
1/1 [==============================] - 0s 2ms/step - loss: 0.2265 - accuracy: 0.9624

[0.22653743624687195, 0.9623529314994812]
```

Figure 32. There is an improvement in the loss value. And the accuracy of the model is 0.9624.

Experiment 2:

```
#from input word get the shape and maximum lenght of the sentence
input_word = Input(shape = (max_len,))
#perform embeddings on input dimension, output dimension, input lenth
model = Embedding(input_dim = num_words,output_dim = max_len,input_length = max_len)(input_word)
#dropout set to 0.1
model = SpatialDropout1D(0.2)(model)
#call Bi-LSTM model and perform NER on model
model = Bidirectional(LSTM(units=25,return_sequences = True, recurrent_dropout = 0.2))(model)
#using timedistributed and softmax activation perform the dense operation
out = TimeDistributed(Dense(num_words_tag,activation = 'softmax'))(model)
#from inpit and output calculate the test parameters
model = Model(input_word,out)
#print the summary of the model
model.summary()
```

```
Model: "model"

_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 50)]              0

embedding (Embedding)       (None, 50, 50)            64600

spatial_dropout1d (SpatialDr (None, 50, 50)           0

bidirectional (Bidirectional (None, 50, 50)           15200

time_distributed (TimeDistri (None, 50, 13)           663
=================================================================
Total params: 80,463
Trainable params: 80,463
Non-trainable params: 0
_________________________________________________________________
```

Figure 33. In second experiment, the values used to tune the model are, dropout= 0.2, learning rate = 25, activation used 'softmax'.

```
# from livelossplot import PlotLossesKeras
#calculate the val_accuracy of the test
early_stopping = EarlyStopping(monitor = 'val_accuracy',patience =2,verbose = 0,mode = 'max',restore_best_weights = Fal:
callbacks = [early_stopping]
#predefining the parameters such as epochs, validation split, verbose parameters
history = model.fit(
    x_train,np.array(y_train),
    validation_split =0.2,
    batch_size = 10,
    epochs = 30,
    verbose =1
)
```

```
Epoch 1/30
13/13 [==============================] - 1s 77ms/step - loss: 2.4148 - accuracy: 0.8284 - val_loss: 2.1699 - val_acc
uracy: 0.9497
Epoch 2/30
13/13 [==============================] - 0s 37ms/step - loss: 1.7566 - accuracy: 0.9484 - val_loss: 0.9223 - val_acc
uracy: 0.9497
Epoch 3/30
13/13 [==============================] - 0s 37ms/step - loss: 0.5221 - accuracy: 0.9484 - val_loss: 0.2970 - val_acc
uracy: 0.9497
Epoch 4/30
13/13 [==============================] - 0s 38ms/step - loss: 0.2911 - accuracy: 0.9484 - val_loss: 0.2668 - val_acc
uracy: 0.9497
Epoch 5/30
13/13 [==============================] - 0s 38ms/step - loss: 0.2757 - accuracy: 0.9484 - val_loss: 0.2556 - val_acc
uracy: 0.9497
Epoch 6/30
13/13 [==============================] - 1s 39ms/step - loss: 0.2611 - accuracy: 0.9484 - val_loss: 0.2454 - val_acc
uracy: 0.9497
Epoch 7/30
13/13 [==============================] - 0s 38ms/step - loss: 0.2476 - accuracy: 0.9484 - val_loss: 0.2382 - val_acc
```

Figure 34. The hyperparameter values used are epochs of 30 and batch size of 10.

4.2 Google Collaboratory

The dataset was manually produced, and a copy of it was shared in the moodle. So, save the files to Google Disk and then mount the drive on the collab laptop. Follow the instructions below to connect your Google Drive to your notebook.

Run the code below in collab to import the dataset into Google Collab.

```
+ Code   + Text                                                                 Connect ▾

select file name "input.txt" by uploading the file..!

[ ] #from google collab import the file name "input.txt", first save in the google drive, then we can read the file
    from google.colab import files
    uploaded = files.upload()

    Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
    Saving input.txt to input.txt
```

Figure 35. Import the file to Google Collaboratory.

Import all the libraries listed in Figure 10.

```
[ ] #import pandas library
    import pandas                as pd
    #import os to get target directory
    import os
    #Encode labels with value between 0 and n_classes-1.
    from sklearn.preprocessing   import LabelEncoder
    #split your dataset into subsets using train test split() from the scikit-learn data science package
    #to reduce the risk of bias in your assessment and validation process.
    from sklearn.model_selection    import train_test_split
    #Score for classification accuracy.
    #This function computes subset accuracy in multilabel classification: the set of labels predicted for
    #a sample must exactly match the corresponding set of labels in y true.
    from sklearn.metrics         import accuracy_score
```

Figure 36. Import the above mentioined librabries.

Install necessary packages for BERT model.

```
!pip install simpletransformers

Collecting simpletransformers
  Downloading simpletransformers-0.61.13-py3-none-any.whl (221 kB)
     |████████████████████████████████| 221 kB 8.2 MB/s
Collecting tensorboardx
  Downloading tensorboardX-2.4-py2.py3-none-any.whl (124 kB)
     |████████████████████████████████| 124 kB 68.2 MB/s
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (from simpletransformers) (2019.12.20)
Collecting datasets
  Downloading datasets-1.11.0-py3-none-any.whl (264 kB)
     |████████████████████████████████| 264 kB 60.5 MB/s
Collecting seqeval
  Downloading seqeval-1.2.2.tar.gz (43 kB)
     |████████████████████████████████| 43 kB 2.5 MB/s
Collecting tqdm>=4.47.0
  Downloading tqdm-4.62.0-py2.py3-none-any.whl (76 kB)
     |████████████████████████████████| 76 kB 5.6 MB/s
Collecting transformers>=4.2.0
  Downloading transformers-4.9.2-py3-none-any.whl (2.6 MB)
     |████████████████████████████████| 2.6 MB 55.6 MB/s
Collecting tokenizers
  Downloading tokenizers-0.10.3-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (3.3 MB)
     |████████████████████████████████| 3.3 MB 51.1 MB/s
Collecting sentencepiece
  Downloading sentencepiece-0.1.96-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.2 MB)
     |████████████████████████████████| 1.2 MB 50.0 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from simpletransformers) (1.19.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from simpletransformers) (1.1.5)
Collecting wandb>=0.10.32
```

Figure 37. Import the simple transformers package.

4.3 Pre-processing the data

Dataset is imported and select only required columns. Search every word for missing values and use method ‘fillna’ to populate the missing values with ‘na’. The input file is converted into ‘utf-8’ format. To understand the data, displaying the top five rows from input.txt.

```
#data is assigned with file name "input.txt" and this file is separated by spaces in between.
#select sentence#, word, POS and Tag frm the input.txt
data = pd.read_csv(os.path.join('input.txt'),delimiter="\t", error_bad_lines=False,
    usecols=['Sentence#', 'Word', 'POS','Tag']
)
index = [i for i in range(0,len(data['Word']))]
#fill na will populate the missing data in the dataset
data = data.fillna(method='ffill')
#printing the shape of data file name.
print(data.shape)
#printing first 5 rows of data fileanem.
data.head()
```

```
(2071, 4)
```

	Sentence#	Word	POS	Tag
0	1.0	ಭಾರತದ	NNP	B-LOC
1	1.0	ಸ್ವಾತಂತ್ರ್ಯ	NN	O
2	1.0	ಚಳುವಳಿಯು	NN	O
3	1.0	ಬ್ರಿಟಿಷ್	NNP	B-COM
4	1.0	ಸಾಮ್ರಾಜ್ಯದಿಂದ	NN	O

Figure 38. Reading the file, populating missing values with ‘na’ and printing top 5 values.

BERT models have standard for naming the variables.

```
#reanaming the column names to default values as per "transfomers"
data.rename(columns={"Sentence#" : "sentence_id", "Word": "words","POS":"pos", "Tag": "labels"}, inplace=True)
```

Figure 39. Renaming the variables to BERT default names.

```
#"Tag" in the data filename is renamed to "labels" and converting all labels to upper
data["labels"] = data["labels"].str.upper()
```

+ Code

Figure 40. Converting the lowercase ‘Tags’ column to uppercase.

```
#x value is assigned with 'sentence#', 'words',and 'pos'
X = data[["sentence_id", "words", "pos"]]
#y is assigned to 'labels'
Y= data["labels"]
```

Figure 41. Assigning dependent and independent variables to X and Y.

```
#splitting the dataset into train dataset and test dataset, with test dataset =0.2 of data filename
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size=0.2)
```

```
[ ] #Assigning the columsn to tain data
train_data = pd.DataFrame({"sentence_id": x_train["sentence_id"], "words":x_train["words"],"pos":x_train["pos"], "labels":y_train})
#Assigning the columns to test data
test_data = pd.DataFrame({"sentence_id":x_test["sentence_id"],"words":x_test["words"], "labels":y_test})
```

Figure 42. Splitting the dataset with test size of 0.2. And assigning the values for test and train data.

```
#to know what all data is populated under train dataset
train_data
```

	sentence_id	words	pos	labels
382	27	ಬಿದನೂರಿನ	I-LOC	B-LOC
725	51	.	O	O
1977	162	ರುದ್ರಗೌಡ	B-PER	B-PER
1935	158	.	O	O
1502	126	ರಾಷ್ಟ್ರೀಯ	I-ORG	I-ORG
...	...	...	...	...
926	71	ಜಾರಿಗೆ	O	O
402	28	ಅಧೀನಪಡಿಸಿಕೊಂಡನು	O	O
1207	97	ಜನರ	B-COM	O
2016	165	ಕಡೆಗೆ	O	O
11	1	ಇದು	PRP	O

Figure 43. Sample data of train data.

```
#calling the NER package of BERT transformers and importing NERmodel and NER arguments
from simpletransformers.ner import NERModel, NERArgs

2021-08-05 15:00:03.792745: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcudart.so.11.0
```

Figure 44. Import NERmodel and NERArgs from simple transformer.

4.4 Model Details

The model code proposed in this part contains the models of segmentation, classification and the result.

4.4.1 BERT Model

The code for implementing the BERT model is shown in Figure 19.

```
[ ] #intialising the pre-run values like epochs, learning rate, overwrite output directory, train batch size and evaluation batch size
    args = NERArgs()
    #after multiple trail and errors, the maximum results acheived with epochs = 3, tried for other multiple values, but epochs= 3 score better
    args.num_train_epochs =3
    args.learning_rate =1e-4
    args.overwrite_output_dir= True
    args.train_batch_size = 32
    args.eval_batch_size = 32
```

Figure 45. Initializing the hyperparameters.

```
#'ber-base-multilingual-cased' which features the support for multiple languages other than English and passing 'labels' and 'arguments' to the NER model
#this package installs the neccessary libraries to train the model on Bert Named Entity Recognition
model = NERModel('bert', 'bert-base-multilingual-cased', labels=label, args=args)

Downloading: 100% 625/625 [00:00<00:00, 17.2kB/s]
Downloading: 100% 714M/714M [00:15<00:00, 44.6MB/s]
Some weights of the model checkpoint at bert-base-multilingual-cased were not used when initializing BertForTokenClassification: ['cls.predictions.decoder.weight', 'cls.predictions.bias', 'cls.seq_r
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassifi
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model f
Some weights of BertForTokenClassification were not initialized from the model checkpoint at bert-base-multilingual-cased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Downloading: 100% 996k/996k [00:00<00:00, 4.37MB/s]
Downloading: 100% 29.0/29.0 [00:00<00:00, 809B/s]
Downloading: 100% 1.96M/1.96M [00:00<00:00, 3.14MB/s]
```

Figure 46. Installing ‘bert-base-multilingual-cased’ NERmodel.

```
[ ] #running the model with epochs=3 and evaluating model with accuracy score
    model.train_model(train_data, eval_data= test_data,acc = accuracy_score)

    100% 1/1 [00:00<00:00, 2.72it/s]
    Epoch 3 of 3: 100% 3/3 [01:29<00:00, 33.40s/it]
    Epochs 0/3. Running Loss: 0.9471: 100% 6/6 [00:01<00:00, 4.08it/s]
    /usr/local/lib/python3.7/dist-packages/simpletransformers/ner/ner_model.py:775: FutureWarning: Non-finite norm encou
      model.parameters(), args.max_grad_norm
    /usr/local/lib/python3.7/dist-packages/torch/optim/lr_scheduler.py:134: UserWarning: Detected call of `lr_scheduler.
      "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate", UserWarning)
    Epochs 1/3. Running Loss: 0.8236: 100% 6/6 [00:01<00:00, 4.14it/s]
    Epochs 2/3. Running Loss: 0.5717: 100% 6/6 [00:01<00:00, 4.14it/s]
    (18, 1.1411700712309942)
```

Add text cell

Figure 47. Evaluating the model for accuracy score.

```
[ ] #the result from the test data
    result

    {'eval_loss': 0.7679847717285156,
     'precision': 0.3617021276595745,
     'recall': 0.17708333333333334,
     'f1_score': 0.2377622377622378}
```

Figure 48. The evaluation result from the test.

5 Visualization

This section gives an overview of the output of the models. The results of the model include the duration of the phrase, types of "tags" and diagrams that indicate the loss of training.

```
#using "ggplot" plot the taotal sentences in the dataset and using histogram plot
#plot the lenth of the sentences
plt.style.use("ggplot")
plt.hist([len(s) for s in sentences], bins=50)
#print the histogram of the column "sentence#"
plt.show()
```

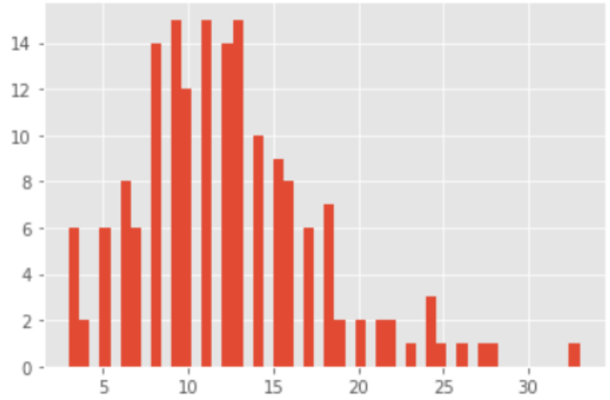

Figure 49. Visualization of the sentence in the dataset.

```
#using seaborn package plot the 'tag' labels..

plt.figure(figsize=(15, 5))
ax = sns.countplot('Tag', data=data)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90, ha="center")
plt.tight_layout()
plt.show()
```

```
C:\Users\vidya\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a
keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

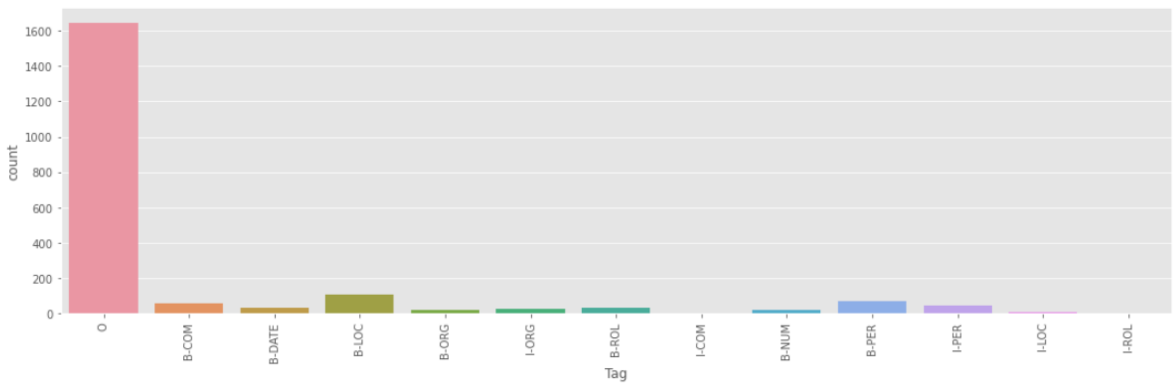

Figure 50. Count plot total number of 'Tags' used in the dataset.

```
#plot the 'tag' with 'O' using seaborn and print the plot
plt.figure(figsize=(15, 5))
ax = sns.countplot('Tag', data=data.loc[data['Tag'] != 'O'])
ax.set_xticklabels(ax.get_xticklabels(), rotation=90, ha="center")
plt.tight_layout()
plt.show()
```

```
C:\Users\vidya\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments with out an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

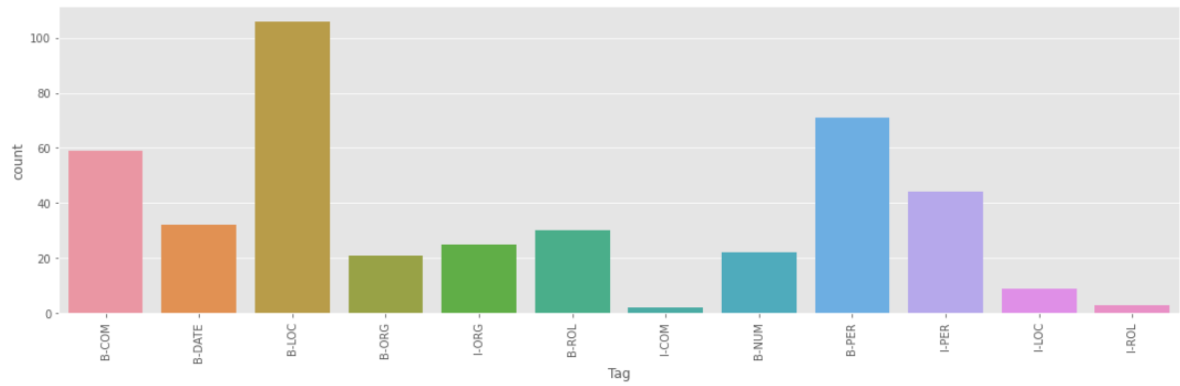

Figure 51. The count plot of 'Tags' with 'o' using seaborn.

```
#using seaborn plot the countplot of 'pos'
plt.figure(figsize=(15, 5))
ax = sns.countplot('POS', data=data, orient='h')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90, ha="center")
plt.tight_layout()
plt.show()
```

```
C:\Users\vidya\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments with out an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

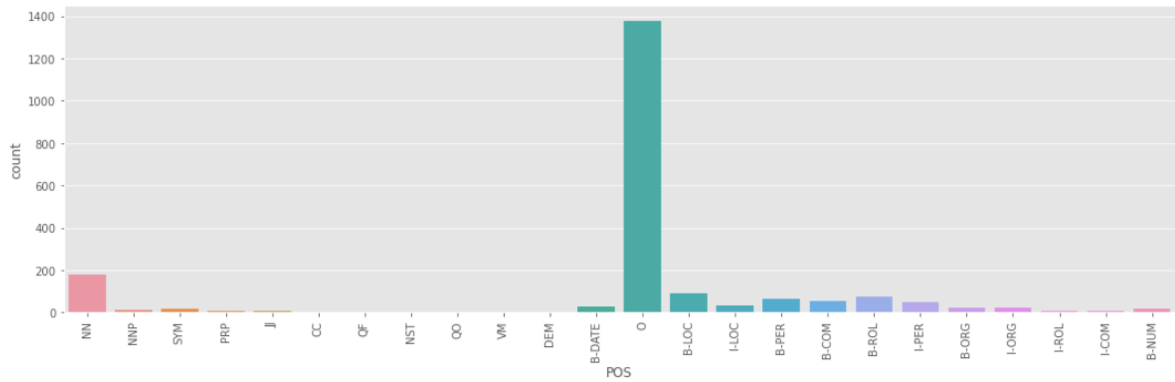

Figure 52. Count plot of 'POS' using seaborn library.

```
#Plot the Loss Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'g',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)
```

```
Text(0.5, 1.0, 'Loss Curves')
```

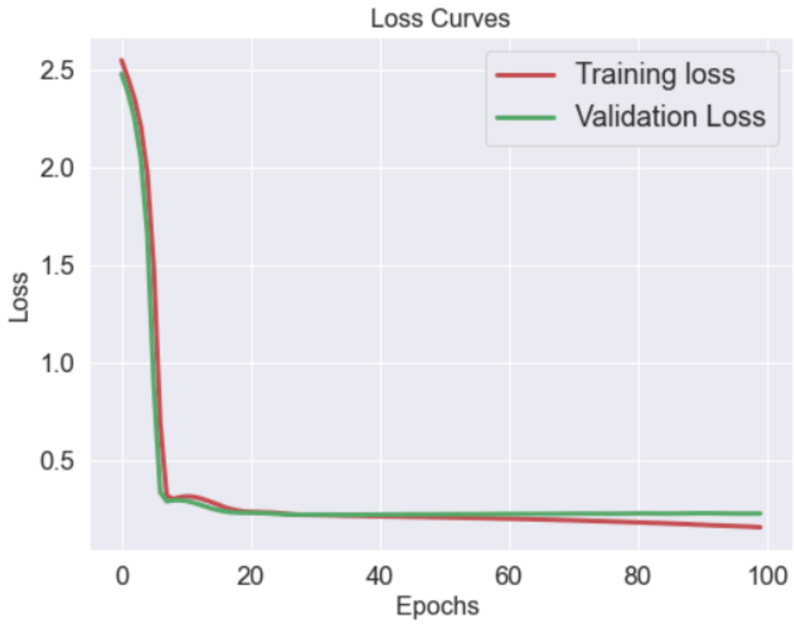

Figure 53. Plot shows the Bi-LSTM model performance based on training and validation loss.

```
#Plot the Loss Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'g',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)
```

```
Text(0.5, 1.0, 'Loss Curves')
```

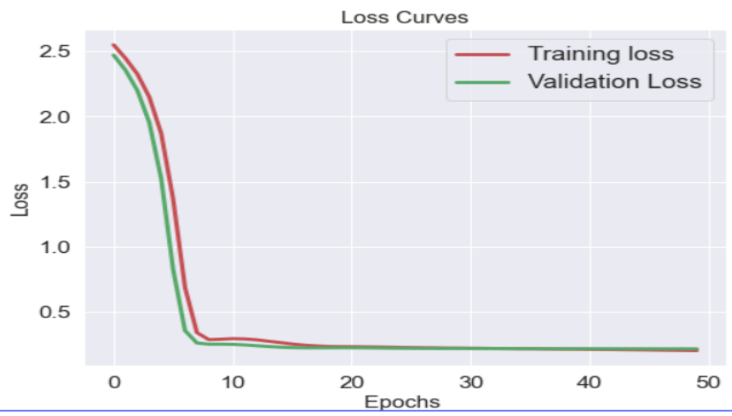

Figure 54. The plot representing the performance of the Bi-LSTM on Kannada Named Entity Recognition.

```
#Plot the Loss Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'g',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)
```

```
Text(0.5, 1.0, 'Loss Curves')
```

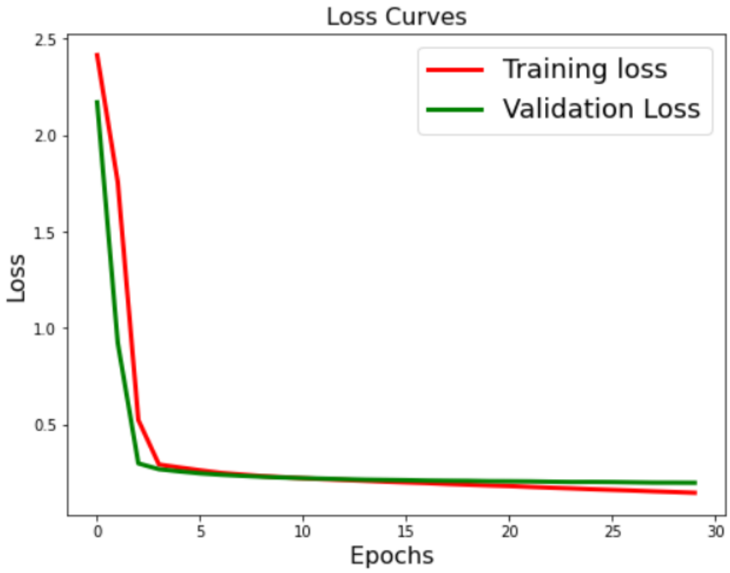

Figure 55. The graph represents the outcome of the experiment 2.