Spam/Ham Text Detection using Dense network, LSTM and Bi-LSTM architectures in TensorFlow2

Purpose

To implement Dense, Long Short Term Memory (LSTM) and Bidirectional-LSTM (Bi-LSTM) deep learning models in TensorFlow2 Keras API for spam/ham texts Detection.

Approach

- Dataset Loading and Exploration.
- Preparing Train and Test Data
- Model training using 3 of the above mentioned deep learning models.
- Choosing the final model with the best Accuracy .
- Using Final trained classifier to classify the new messages.

Spam Data Exploration

Libraries for

- Reading data,exploring and plotting.
- Train test split
- Text preprocessing of Deep Learning
- Modelling

Text Dataset is a tab separated (\t) text file.

Statistics summary

- ☐ 5,572 labels and messages
- ☐ two unique labels indicating for “ham” and “spam”
- ☐ Lesser unique messages (5,169) than total message count(5,572) indicating some repeated messages.
- ☐ duplicates = df[df.duplicated()],shows there are 403 duplicated messages.
- ☐ The top label and top message are “ham” and “Sorry, I’ll call later” respectively.
- ☐ Data is imbalanced as the number of ham is 4,825 compared to 747 spam messages.
- ☐ Popular ham sms = “Sorry, I’ll call later” and Popular spam sms = “Please call our customer service…”
- ☐ On average, the ham message has length of 73 words whereas spam message has 138.

WordCloud and Bar charts Visualisation

- ☐ Create a separate data frame for ham and spam texts and convert it to a numpy array to generate WordCloud.
- ☐ Extract words most commonly found in ham and spam messages, remove meaningless stop words such as “the”, “a” , “is” etc, and plot it.
- ☐ According to the Bar plot ,there are more frequent ham messages (85%) than spam (15%).

Ways to handle imbalanced data

- Using appropriate evaluation metrics .
- Resampling the Dataset (oversampling/upsampling or undersampling/downsampling)
- Putting different resampled Datasets together.

Downsampling the Majority class(Ham)

After Downsampling , 747 messages in each class.

Prepare train/test data and pre-process text

- 80% of data were used for training and 20% for testing purposes.
- Convert labels to numpy arrays to fit deep learning models.
- Text pre-processing which includes Tokenization, Sequencing and Padding.

Compare three different models and select a final one

```
print(f"Dense architecture loss and accuracy: {dense_model.evaluate(test_pad, test_label)} " )
print(f"LSTM architecture loss and accuracy: {lstm_model.evaluate(test_pad, test_label)} " )
print(f"Bi-LSTM architecture loss and accuracy: {Bilstm_model.evaluate(test_pad, test_label)} " )
```

```
10/10 [==============================] - 0s 667us/step - loss: 0.1132 - accuracy: 0.9465
Dense architecture loss and accuracy: [0.11318477243185043, 0.9464883208274841]
10/10 [==============================] - 0s 4ms/step - loss: 0.2466 - accuracy: 0.9294
LSTM architecture loss and accuracy: [0.24663574993610382, 0.9293644428253174]
10/10 [==============================] - 1s 4ms/step - loss: 0.6891 - accuracy: 0.5543
Bi-LSTM architecture loss and accuracy: [0.6891114115715027, 0.554314374923706]
```

The dense spam detection model outperformed other two models in terms of Accuracy .Hence,this will be trained and evaluated.

Case 1: Given the text from our original data(Raw text from given dataset)

```
predict_sms_ham_spam = ['Sorry, I'll call later']
```

```
def predict_spam(predict_sms_ham_spam):
    new_seq = tknizer.texts_to_sequences(predict_sms_ham_spam)
    padded = pad_sequences(new_seq, maxlen =max_len,
                      padding = pad_type,
                      truncating=trunc_type)
    return (dense_model.predict(padded))
predict_spam(predict_sms_ham_spam)
```

```
array([[0.0255329]], dtype=float32)
```

```
final = predict_spam(predict_sms_ham_spam)
def spam_or_ham(final):
    for message in final:
        if message>=0.7:
            print("SPAM")
        else:
            print("NOT SPAM")
        message = message+1
spam_or_ham(final)
```

```
NOT SPAM
```

Case 2: Given the New data

```
predict_sms_ham_spam = ["You are awarded a Nikon Digital Camera. Call now"]
```

```
def predict_spam(predict_sms_ham_spam):
    new_seq = tknizer.texts_to_sequences(predict_sms_ham_spam)
    padded = pad_sequences(new_seq, maxlen =max_len,
                      padding = pad_type,
                      truncating=trunc_type)
    return (dense_model.predict(padded))
predict_spam(predict_sms_ham_spam)
```

```
array([[0.94411016]], dtype=float32)
```

```
final = predict_spam(predict_sms_ham_spam)
def spam_or_ham(final):
    for message in final:
        if message>=0.7:
            print("SPAM")
        else:
            print("NOT SPAM")
        message = message+1
spam_or_ham(final)
```

```
SPAM
```

Dense spam model correctly classifies the first case as Ham and second case as Spam .

Improvement

Trying

- ☐ more sampling approaches like upsampling ,SMOTE etc
- ☐ using different hyper-parameters.
- ☐ sample size increment.
- ☐ Machine Learning Classifiers